21世纪高等学校规划教材 | 计算机应用



ASP.NET网站开发项目化教程

肖宏启　编著

清华大学出版社

21世纪高等学校规划教材·计算机应用

ASP.NET网站开发项目化教程

肖宏启　编著

清华大学出版社
北京

内 容 简 介

本书以“新知书店”网站项目的开发过程为主线，以 C# 为编程语言，讲述了 Web 应用程序开发从系统架构到编码实现的过程。

全书共分为 10 个单元，包括网上书店项目需求分析与设计、ASP.NET 基础及开发环境构建、使用控件高效创建网站页面、系统对象与数据传递、搭建风格统一的 Web 站点、数据库访问及网上书店系统架构、数据绑定技术、数据绑定控件的应用、数据绑定控件应用进阶、“新知书店”购物功能的设计与实现。

全书结合专业课程特点，对基于 ASP.NET Web 软件开发工作过程进行剖析，以真实完整的项目“新知书店”为载体，在行业专家的指导下，结合 Web 项目开发的流程和规范，分解出工作过程的典型工作任务，根据工作任务整合相关知识点，按照应用型本科及高职学生的认知特点设计教学过程，把基础知识的应用渗透到各个项目任务中。任务讲解步骤清晰，循序渐进，通过对项目任务的学习，读者可以更好地领会 ASP.NET 的语法和编程技巧，有助于将所学的知识融会贯通。

本书内容丰富，层次清晰，讲解深入浅出，可作为高等院校应用型本科、专科及高等职业院校计算机类专业 Web 应用程序开发课程的教材，也可作为培训班的培训教材，还可供从事 ASP.NET 开发和应用的相关人员学习与参考

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目（CIP）数据

ASP.NET 网站开发项目化教程/肖宏启编著.--北京：清华大学出版社，2015
21 世纪高等学校规划教材·计算机应用
ISBN 978-7-302-37935-5

Ⅰ.①A…　Ⅱ.①肖…　Ⅲ.①网页制作工具－程序设计－高等学校－教材　Ⅳ.①TP393.092

中国版本图书馆 CIP 数据核字（2014）第 207784 号

责任编辑：黄　芝　李　晔
封面设计：
责任校对：时翠兰
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦 A 座　　**邮　　编**：100084
社 总 机：010-62770175　　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　　**印　　张**：24　　**字　　数**：583 千字
版　　次：2015 年 1 月第 1 版　　**印　　次**：2015 年 1 月第1 次印刷
印　　数：1～　000
定　　价：　.00 元

产品编号：060197-01

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施“高等学校本科教学质量与教学改革工程(简称‘质量工程’)”，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部“质量工程”的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合“质量工程”的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的“高等学校教学质量与教学改革工程精品课程建设工作”，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了“清华大学出版社教材编审委员会”(以下简称“编委会”)，旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。“编委会”成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，“编委会”一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上；精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性(新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的“编委会”成员推荐)，经“编委会”认真评审，最后由清华大学出版

社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括：

（1）21世纪高等学校规划教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算机应用类教材。

（2）21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

（3）21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

（4）21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

（5）21世纪高等学校规划教材·信息管理与信息系统。

（6）21世纪高等学校规划教材·财经管理与应用。

（7）21世纪高等学校规划教材·电子商务。

（8）21世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人：魏江江
E-mail：weijj@tup.tsinghua.edu.cn



前　言

随着社会经济的高速发展，我国的高等教育已从精英教育走向大众化教育的发展阶段。国际高等教育的历史说明高等职业教育必将成为这一发展过程的主力军。对于我国这种从专科转型的高等职业教育，现今还处于探索阶段，教材建设是教育改革的重要方面。

高等职业教育是高等教育的一种类型，它与传统的普通高等教育既联系紧密，又有本质的区别，高等职业教育强调面向社会、生产、管理、服务第一线，培养技术应用性人才，学生毕业后，即可发挥其所学专长。所以，根据我国教育部规定，高等职业教育需根据其自身的特点，建立自己的教材体系。

本书是项目化人才培养的创新教程，突出以工作过程为导向，以工作任务为基础，突出职业和实践特色，侧重培养学生软件设计、代码编写、软件文档编写规范等能力。

开发网站的基本要求是：前台页面美观，后台数据安全、开发效率高、数据访问速度快。ASP.NET 开发技术在这些方面都有一定的优势，也深受 Web 开发人员的青睐。作为一本 ASP.NET 开发技术的入门教材，本书不是简单地罗列控件和对象，也不是浅显地介绍数据访问方面的知识，而是从网站开发的实际需求和学习者的认知规律出发设计教学单元和操作任务。本书共设计了 10 个教学单元和 38 项操作任务，10 个教学单元的排列顺序为：项目需求分析与设计—开发平台搭建—定制界面—应用对象—设计外观与导航—数据访问与系统架构—数据绑定—数据查询、展示与编辑—用户管理—购物车的设计与实现。38 项操作任务合理地分布在各个教学单元中，这些任务都是按照由易到难、由简单到复杂的顺序设置的。

本书特色：

(1) 面向企业实际需求，以培养能力为目标。本书面向企业的实际工作需要，以企业实际应用项目为素材，以积累项目经验、培养应用能力、形成企业工作规范为目标，按照项目开发的工作过程组织编写。

(2) 以项目为载体，以任务为驱动。全书以“新知书店”网站开发项目为载体，将学习过程、工作过程与学生知识技能的培养联系起来，构建 10 个单元，每个单元分解为若干任务。

(3) 配套资源丰富。免费提供包括素材、源文件、教学课件、课程标准、课程整体教学设计、单元教学设计、考核方式的教学资源，可在清华大学出版社的网站下载，网址为 http://www.tup.tsinghua.edu.cn。

本教程主要面向应用型本科及高等职业院校计算机类专业的学生，内容构造体现“以应用为主体”，强调知识的理解和运用，实现高校应用型本科及高等职业教育教学以实践体系为主及以技术应用能力培养为主的目标，符合现代高等职业教育对教材的需求。

本书由肖宏启编著，参与资料整理和程序调试的有贵州航天职业技术学院软件技术重点专业建设项目组成员陈美成、汤智华、苏畅、柳均、陆树芬、周奇衡等老师。本书编写过程中得到了院长孙平安，副院长王逸群、冯伟，信息工程系系主任杨先立的大力支持，在此对大

家的辛勤工作表示衷心感谢。本书在编写过程中，还参考了近5年出版的ASP.NET技术相关专著、教材及杂志，以及互联网上的相关资料，在此一并表示衷心的感谢。最后，感谢所有在本书写作过程中给予帮助的人，特别是在此过程中默默付出的我的妻子燕雁。

本教材的结构是一种新的尝试，能否得到同行的认可，能否给教学带来新的感受，需要经过实践的检验。特别希望各位读者能通过作者邮箱 xiaohongqi2000@163.com 分享体会，提供意见建议。由于编写教材的时间紧张，难免存在疏漏，敬请读者批评指正。

编　者

2014年10月

目 录

单元1 网上书店项目需求分析与设计

教学目标：

- 学会根据客户描述分析业务需求。
- 掌握根据业务需求进行系统分析与设计，确定功能模块，画出流程图的方法。
- 掌握根据需求设计数据库并创建表的方法。

1.1 知识准备

1.1.1 需求分析

本书以“新知书店”这一项目贯穿始终，在软件开发项目中，需求分析是一个非常重要的环节，可以说需求分析与设计是否合理决定了软件开发项目实施的成败。软件开发项目的需求分析主要解决做什么的问题，设计是解决如何做的问题。“新知书店”这一软件开发项目的需求分析包括需求调研、功能与性能分析、编写需求分析报告、需求评审；设计包括框架设计、数据库设计，详细设计将在后面各单元中分别展开。

1. 需求分析的定义

IEEE 软件工程标准词汇表(1997 年)中定义的需求为：

(1) 用户解决问题或达到目标所需的条件或能力。

(2) 系统或系统部件要满足合同、标准、规范或其他正式规定文档所需具有的条件或能力。

(3) 一种反映上述条件和能力的文档说明。

2. 需求分析的任务

需求分析的任务是借助于当前系统的物理模型(待开发系统的系统元素)导出目标系统的逻辑模型(只描述系统要完成的功能和要处理的数据)，解决目标系统“做什么”的问题，所要做的工作是深入描述软件的功能和性能，确定软件设计的限制和软件同其他系统元素的接口细节，定义软件的其他有效性需求，通过逐步细化对软件的要求描述软件要处理的数据，并给软件开发提供一种可以转化为数据设计、结构设计和过程设计的数据与功能表示。必须全面理解用户的各项要求，但不能全盘接受，只能接受合理的要求；对其中模糊的要求

要进一步澄清,然后决定是否采纳;对于无法实现的要求要向用户作充分的解释。最后将软件的需求准确地表达出来,形成软件需求说明书(Software Requirements Specification,SRS),其实现步骤如图 1-1 所示。

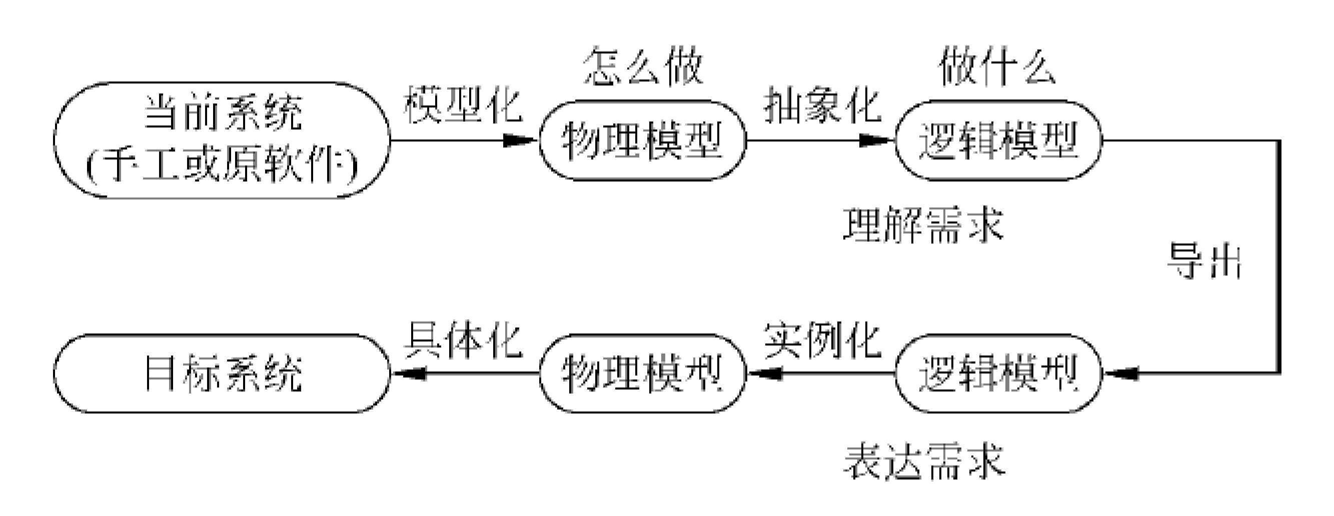


图 1-1 软件开发项目需求分析实现步骤

(1) 获得当前系统的物理模型:首先分析、理解当前系统是如何运行的,了解当前系统的组织机构、输入输出、资源利用情况和日常数据处理过程,并用一个具体的模型来反映自己对当前系统的理解。此步骤也可以称为“业务建模”,其主要任务是对用户的组织机构或企业进行评估,理解其需要及未来系统要解决的问题,然后建立一个业务 USECASE 模型和业务对象模型。当然如果系统相对简单,也没必要大动干戈去进行业务建模,只要做一些简单的业务分析即可。

(2) 抽象出当前系统的逻辑模型:在理解当前系统“怎样做”的基础上,去除非本质因素,抽取出“做什么”的本质。

(3) 建立目标系统的逻辑模型:明确目标系统要“做什么”。

(4) 对逻辑模型的补充,如用户界面、启动和结束、出错处理、系统输入输出、系统性能、其他限制,等等。

3. 需求分析的过程

需求分析可分为问题识别、分析与综合、编制需求分析文档和需求评审四个阶段,需求调研属于问题识别阶段。需求分析个过程如下:

(1) 问题识别。解决目标系统做什么,做到什么程度。需求包括功能、性能、环境、可靠性、安全性、保密性、用户界面、资源使用、成本、进度。同时建立需求调查分析所需的通信途径。

(2) 分析与综合。从数据流和数据结构出发,逐步细化所有的软件功能,找出各元素之间的联系、接口特性和设计上的限制,分析它们是否满足功能要求并剔除不合理部分,综合成系统解决方案,给出目标系统的详细逻辑模型。常用的分析方法有面向数据流的结构化分析方法 SA(数据流图 DFD、数据词典 DD、加工逻辑说明)、描绘系统数据关系的实体关系图 ERD、面向数据结构的 Jackson 方法 JSD、面向对象分析方法 OOA(主要用 UML)、对于有动态时序问题的软件可以用形式化技术,包括有穷状态机 FSM 的状态迁移(转换)图 STD、时序图、Petri。每一种分析建模方法都有其优势和局限性,可以兼而有之,以不同角度分析,应该避免陷入在软件需求方法和模型中发生教条的思维模式和派系斗争,一般来说结构化方法用于中小规模软件、面向对象方法用于大型软件。

(3) 编制需求分析文档。得到一个目标系统的蓝图。

(4) 需求评审。通过与用户及有该系统相关经验的专家进行评审,得到用户确认后才可以进行下一步的软件设计工作。

软件开发项目中,没有正确的客户需求比没有需求更可怕。需求分析不仅仅是单方面记录用户的需求,而是通过与用户通力合作,与客户一起探索需求,并达成对问题的共识,以及合作探索出最佳解决方案。

4. 需求调研方法

要得到有效的用户需求,需要进行需求调研,收集相关信息,具体方法有:实地操作、访谈、特殊群体调查、问卷调查、情景分析、原型等。

1) 实地操作

到用户的实际工作环境中观察用户如何完成任务,并向用户询问与任务相关的问题,这样可以得到第一手真实信息,还可以了解到完成一个具体任务的目的,要尽可能地多收集信息,还要鼓励用户多解释完成该任务的原因,观察用户操作和听取用户解释是被动的,而询问用户则是主动的。

2) 访谈

访谈是需求分析人员与用户进行会谈,访谈者和被访谈者的技能决定了访谈收集息的质量,访谈在收集某些信息时非常有用,如跨多个单独任务或部门的过程。

3) 问卷调查

由一组用来收集信息的问题组成一张问卷,进行匿名调查,要得到最有用的信息,需要专业人员来设计,并分析问卷调查结果,通过问卷调查,得到经过仔细考虑的书面回答可能比会谈中的回答更加准确,不过问卷的结果可能会受用户态度影响,具有主观性。

4) 情景分析

利用情景分析诱导用户能够把需求告诉分析员(可以描述当前一项业务怎么做,也可以描述设想的系统中此项业务怎么做)收集分析客户使用的各种表格、有关工作责任、工作流程、工作规范、相关数据标准、业务标准的各种文字资料,从中得到一些要处理的数据和过程。

5) 原型

能够模拟创建被提议的解决方案来收集信息,当在一个常规工作环境中不可能实地操作时,原型就非常有用,可以收集同类相关产品的宣传资料、技术资料、演示程序或类似的软件程序。原型法能帮助验证或记录从用户和业务的角度得出的信息。但如果自己建立原型,则成本比较高。

5. 需求调研基本策略

(1) 首先确定用户的软件开发目标,确定系统基本范围,然后围绕这一目标,确定要访谈的部门和人员、要了解的业务,在基本范围内展开调研。

(2) 以部门职责为基础搞清各种现有业务、要填写的各类表格、文档和生成的报表等及其数据来源和去向。

(3) 以业务为主线,搞清每个业务环节的实现流程、涉及人员、输入输出项。

(4) 以数据为主线,搞清数据采集方式,数据流向、数据之间的内在联系。

(5) 如果用户有其他已建系统不能继续使用或替换,则要搞清哪些业务或数据是已建

系统的,并要弄清已建系统的实现流程和管理数据。

(6) 应思考是否有新技术和新流程可以改进现有业务系统,用户提出的需求能否用现有技术在经济允许的条件下实现。

6. 需求分析组成

软件开发项目需求分析中,一般可以从三个方面去考虑。

1) 功能需求

产品应该完成哪些功能,即向用户提供的功能,一般来说,这个都是比较硬性的标准。

2) 非功能性需求

用户可能不能明确地提出一些需求,比如说性能达到什么标准,可靠性、响应时间、扩展性、性能等方面的要求。

3) 一些约束

在需求分析中需要考虑一些条件约束、规则等,比如客户的约束、行业的约束、法律的约束以及自我的约束等。

软件需求的各组成员部分如图 1-2 所示。

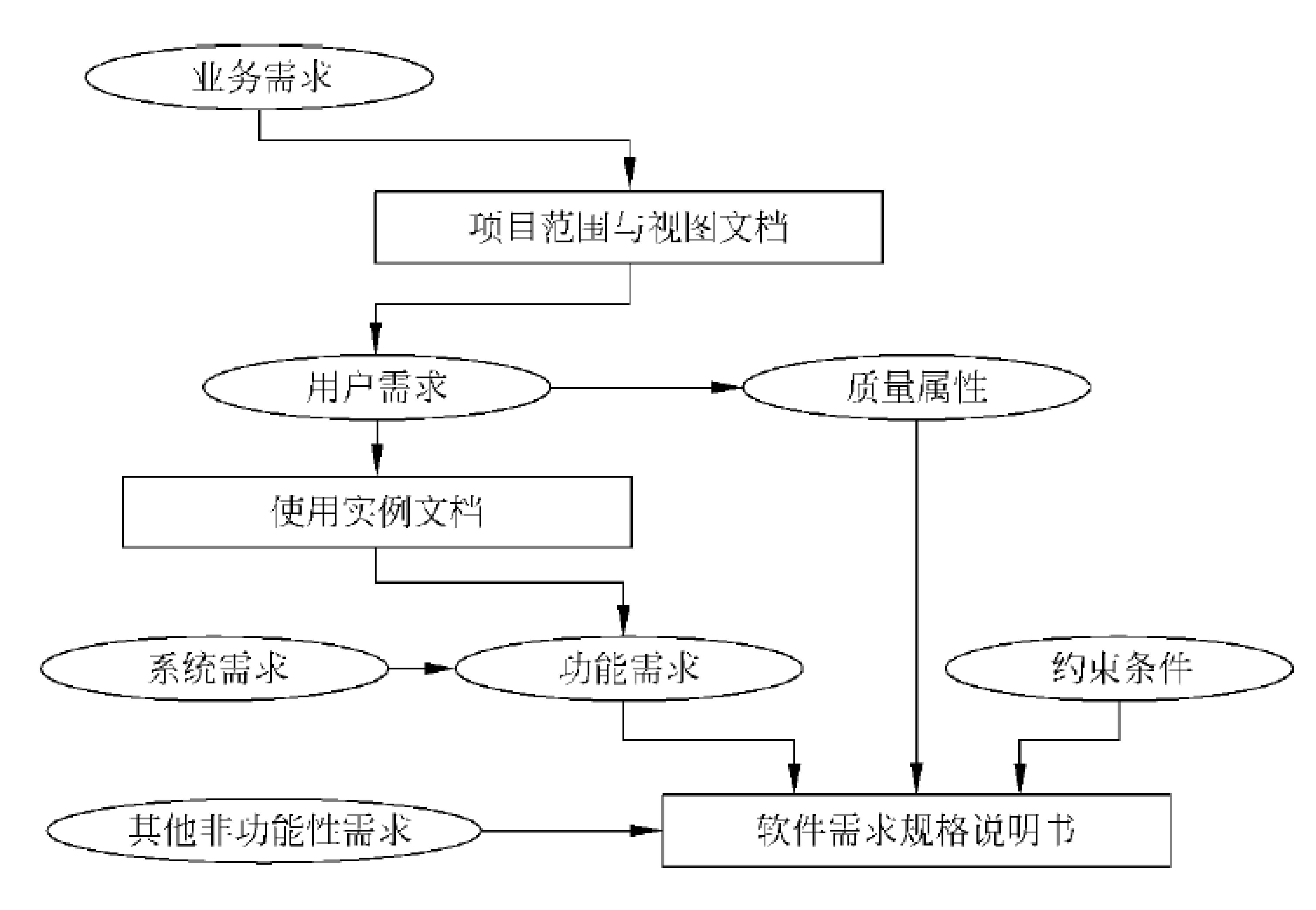


图 1-2 软件开发项目需求的组成

7. 需求文档规范

需求文档的编写需要使用多种软件工具,调研结果"需求分析说明书"的格式参照开发文档模板,一般使用 Word 和 Excel 实现文本和表格的编写。而功能模块分解图、组织结构图可以用 Visio 或 Word 绘制,面向对象的 UML 分析设计图可以用 Visio 或 Rational Rose 绘制,数据物理模型用 PowerDesigner 或 QDesigner 绘制。

1) 编写方法

以把需求说清楚为目的,可以由自然语言描述文本,以图形化的模型来描绘系统状态、变化及对象类关系等,有时还可以通过使用精确的形式化逻辑语言来定需求。

2）应有成果

（1）各业务手工办理流程文字说明。

（2）各业务手工办理流程图。

（3）各业务手工办理各环节输入输出表单、数据来源。

（4）目标软件系统功能结构图及文字说明。

（5）目标软件系统中的各业务模块处理流程模型及文字说明。

（6）目标软件系统中的各业务模块处理数据、数据采集方式、数据间的内在联系分析。

（7）目标软件系统用户界面设计图，系统各模块逻辑联系图及说明。

3）需求文档编写原则

（1）句子简短完整，具有正确的语法、拼写和标点。

（2）使用的术语与词汇表中所定义的一致。

（3）需求陈述应该有一致的样式，例如“系统必须……”或者“用户必须……”，并紧跟一个行为动作和可观察的结果。

（4）避免使用模糊、主观的术语，减少不确定性，如“界面友好、操作方便”。

（5）避免使用比较性词语，如“提高”，应定量说明提高程度。

1.1.2 数据库设计

1. 数据库设计概述

目前，数据管理技术已离不开数据管理系统（DDMS），通过数据库管理系统，可以实现数据的充分共享、交叉访问与应用程序的高度独立，同时数据管理理系统对数据的完整性、唯一性和安全性提供了一套有效的管理手段，另外，还提供了管理和控制数据的各种操作命令，便于编程应用。

对于数据库的设计，一般从概念模型开始，数据模型是对现实世界特征数据的模拟和抽象，在概念模型设计阶段，着重分析数据所的逻辑结构，避免陷入具体的存储细节，所有的设计都与将来所要采用的具体数据库有关。

要开发一个基于数据库的应用系统，其中最关键的一步就是整个系统所依据的数据库的建模设计，从逻辑的到物理的，一个环节疏于设计，整个应用系统便似建立在危房之上，同时，随着开发过程的不断深入，还要随时面对各种难以预料的风险，开发者要为修改或重新设计没有设计好的数据库系统而付出难以预料的代价，所以，一个良好的数据库设计是高效率的系统所必需的。

2. 逻辑建模

数据库设计的方法因具体数据而异，但是建模阶段是相同的，所以可以用一些通用工具来进行，如 Rational rose、PowerDesigner 等，这一阶段主要是依据系统的需求，获取与分析要实现的应用系统信息，进行数据内部以及外在关系的分析，从而有效地建立整个系统的数据结构（在关系数据库中通常称为表结构），在此基础上对数据库的数据量、数据流量及响应速度估算分析，这样数据模型就产生了。具体的操作准则是数据库的几个范式、用户的具体需求和分析者的经验，以及从数据库的性能、安全、方便管理、易于开发等方面出发；具体方

法可以因分析员的喜好和习惯而异，可以不用工具，但最好使用工具，因为这能使分析过程简便，最主要的是可以生成一些图，如E-R图，让分析过程一目了然。

3. 物理设计

此步设计和系统将具体使用的数据库有关，也和数据库所运行的硬、软件平台有关，目的是尽量合理地给数据库分配物理空间。这一步在数据库设计中很重要，关系到数据库数据的安全和数据库的性能，具体来说，这一步包括相应表空间的数据文件在磁盘上的分配，还要根据数据量的大小确定重做日志文件、回滚段的大小，然后进行分配，这些文件的分配要遵循利于备份、利于性能优化的原则。

4. 数据库设计原则

在定义数据库表和字段需求(输入)时，首先应检查现有的或者已经设计出的报表、查询和视图(输出)来决定为了支持这些输出哪些是必要的表和字段。E-R图表和数据字典可以让任何了解数据库的人都明确如何从数据库中获得数据。E-R图为说明表之间关系很有用，而数据字典则说明了每个字段的用途以及任何可能存在的别名。对SQL表达式的文档化来说这是完全必要的。另外，数据库各种对象的命名也必须符合规范。

(1) 标准化和规范化。数据的标准化有助于消除数据库中的数据冗余。标准化有好几种形式，但Third Normal Form(3NF)通常被认为在性能、扩展性和数据完整性方面达到了最好平衡。简单来说，遵守3NF标准的数据库的表设计原则，即某个表只包括其本身基本的属性，当不是它们本身所具有的属性时需进行分解。表之间的关系通过外键相连接。它的特点是有一组表专门存放通过键连接起来的关联数据。

(2) 数据驱动。采用数据驱动而非硬编码的方式，许多策略变更和维护都会方便得多，大大增强了系统的灵活性和扩展性。

1.2 单元任务

任务1-2-1 “新知书店”项目需求分析

【任务描述】

“新知书店”项目系统参照成熟的商业网站，如当当网、卓越亚马逊网等，采用B/S架构，有多个功能模块，分为前台和后台两部分：前台包括图书展示和销售(图书类别列表、图书详细信息显示、图书搜索、购物车管理、订单生成与付款等)、网站用户中心(客户登录、会员资料修改、收藏夹、图书评论等)、首页与图书推荐、其他辅助信息发布等功能模块；后台包括用户信息管理、订单管理、图书类别与详细信息管理、采购与库存管理、配送管理、财务管理、系统管理等功能模块。考虑到有的业务需要特定条件才能实现，另外限于篇幅因素，这里采用简单化处理，比如付款、采购、库存、配送、财务等模块在本系统中就不实现了，只选取了主要功能模块来进行实现。

根据需求分析要求，实现以下任务：

- 设计网站总体结构。

- 设计系统技术框架。
- 设计主要实现流程。
- 数据字典。
- 其他内容。

【任务实施】

根据系统确定的边界范围，确定“新知书店”的功能模块，采用 Visio 或 Word 绘制总体结构图、系统技术框架图、模块功能的用例图和前后台的实现流程，完成相关文档，如需求调研报告、用例提取及描述、数据字典等。

1. 网站总体结构设计

总体结构设计的主要任务是将整个系统合理的划分为多个功能模块，正确地处理模板之间与模块内部的联系以及它们之间的调用关系和数据联系，并定义整个模板的内部结构。通过调查了解及对商业网站的分析，“新知书店”项目分为前台系统和后台管理系统，前台系统分为网站首页、图书列表、图书展示、图书搜索、购物车管理、会员中心、订单生成、用户登录、资料修改、发布评论等功能模块，而后台管理系统分为订单管理、类别管理、图书信息管理、用户管理、系统管理、图书推荐、通知发布等功能模块。“新知书店”总体结构如图 1-3 所示。

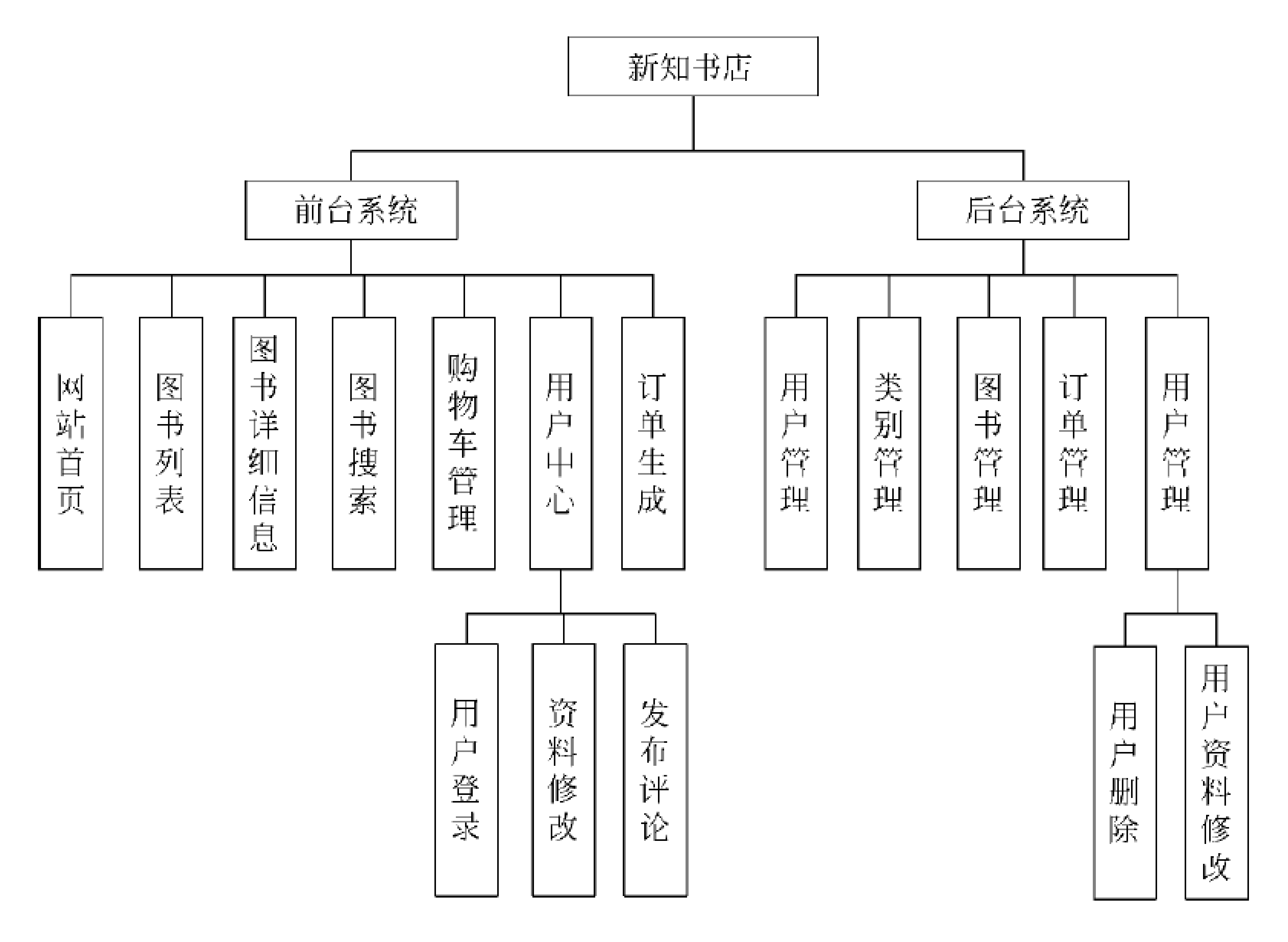


图 1-3 “新知书店”系统总体结构图

2. 网站主要用例描述与功能流程

“新知书店”项目主要从用户和网站管理员的角度出发来进行，系统参与者有游客、会员和管理员 3 种用户。游客可以浏览图书信息，没有注册为会员，所以不能添加图书到购物车、下订单、购买图书等；会员是注册了的用户角色，所以能够浏览图书、添加图书到购物

车、下订单、购买图书，也能够修改自己的信息、购物车的信息和订单的信息等功能；管理员具有后台管理的所有功能，包括前台功能。系统分前台和后台两个主要流程来进行，前台为客户网上购书流程，后台为后台系统管理应用。

（1）确定“新知书店”系统的参与者后，必须确定参与者所使用的用例，用例是参与者与系统交互过程中需要系统完成的任务。识别用例的最好方法是从参与者的角度开始分析，这一过程可通过提出“要系统做什么？”这样的问题来完成。由于系统中存在 3 种类型的参与者，经过详细分析，从这 3 种类型的参与者角度出发，系统具有如表 1-1 所示的用例。

表 1-1 “新知书店”网站系统用例列表

用例名称	描述
登录	会员登录系统
注册	游客注册成为会员
查看新书预览	BOOKSHOP 中的新增书籍
浏览购物车物品	查询当前的购买物品信息
搜索图书	查询需要购买的图书
图书高级搜索	按一个或多个信息查询图书
浏览图书列表	显示查询的书籍列表
浏览图书详细信息	查看图书的详细信息
分类查看图书	按类别查看书籍
购买书籍	将需要购买的书籍放入购物车
浏览公告	查看当前 BOOKSHOP 系统公告
浏览广告	显示，系统当前轮换图片中的广告
支付订单	结算，跳出并登录支付宝页面
修改密码	会员、管理员进行密码修改
查询订单	查看历史订单信息
收货信息设置	设置收货地址、邮编等详细信息
订单查看或修改	查看、修改订单信息
维护公告、广告	管理员添加、修改、删除等广告管理
维护用户信息	管理员对会员进行维护
维护图书信息	管理员对图书进行维护

限于篇幅，这里只选取几个典型的用例进行描述，需要说明的是在后续单元中并非所有用例所描述的功能都实现，只作为需求分析列出，供有兴趣的读者扩展。

（2）“新知书店”系统前台的程序流程图如图 1-4 所示。

（3）浏览图书列表用例分析与描述。

浏览图书列表用例图如图 1-5 所示。

浏览图书列表用例描述如表 1-2 所示。

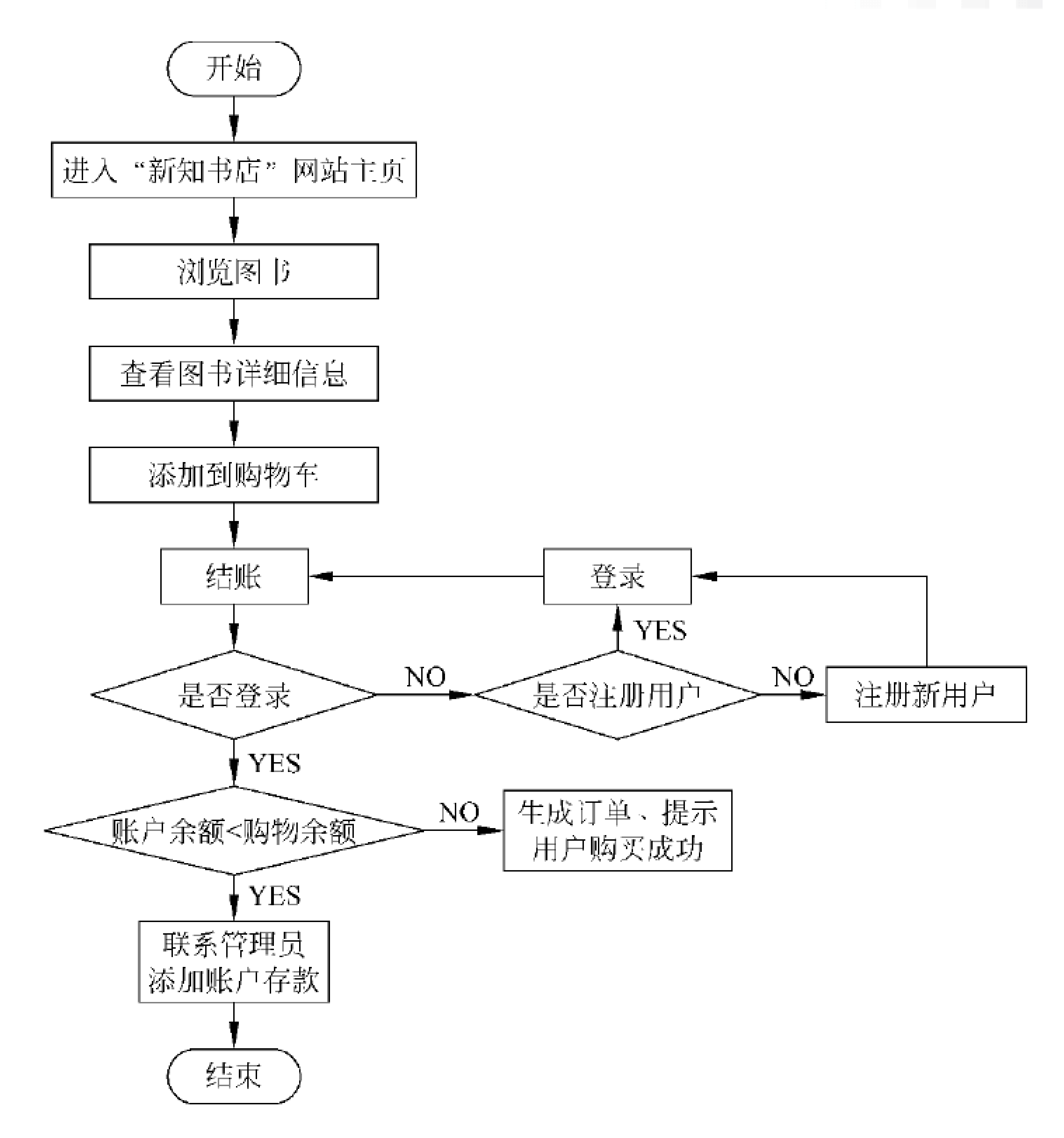


图 1-4　"新知书店"系统前台的程序流程图

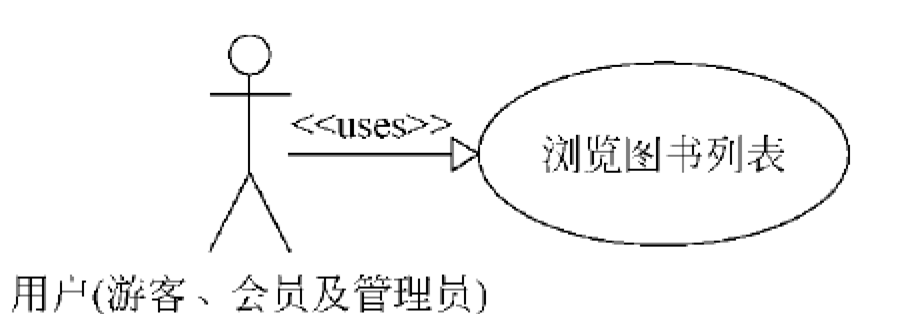


图 1-5　浏览图书列表用例

表 1-2　浏览图书列表用例的细化描述

用例名称	浏览图书列表
用例描述	显示查询的书籍列表
参与者	图书管理员、会员用户及游客(所有顾客)
基本操作流程	① 用户单击 Default.aspx 或 BookList.aspx 页面中的某图书类别 ② 系统显示该类别的子类别。该过程一直持续下去,直到没有子类别为止,此时系统将显示最小子类别中的图书 ③ 用户单击某本图书的封面或标题。系统调用"浏览图书详细信息"用例
可选操作流程	如果系统在指定的类别中没有找到任何图书,则跳转到首页 Default.aspx,以指出这一点并提示顾客选择其他类别

浏览图书列表交互页面如图 1-6 所示。

(4)登录用例分析与描述。

会员用户登录用例图如图 1-7 所示。

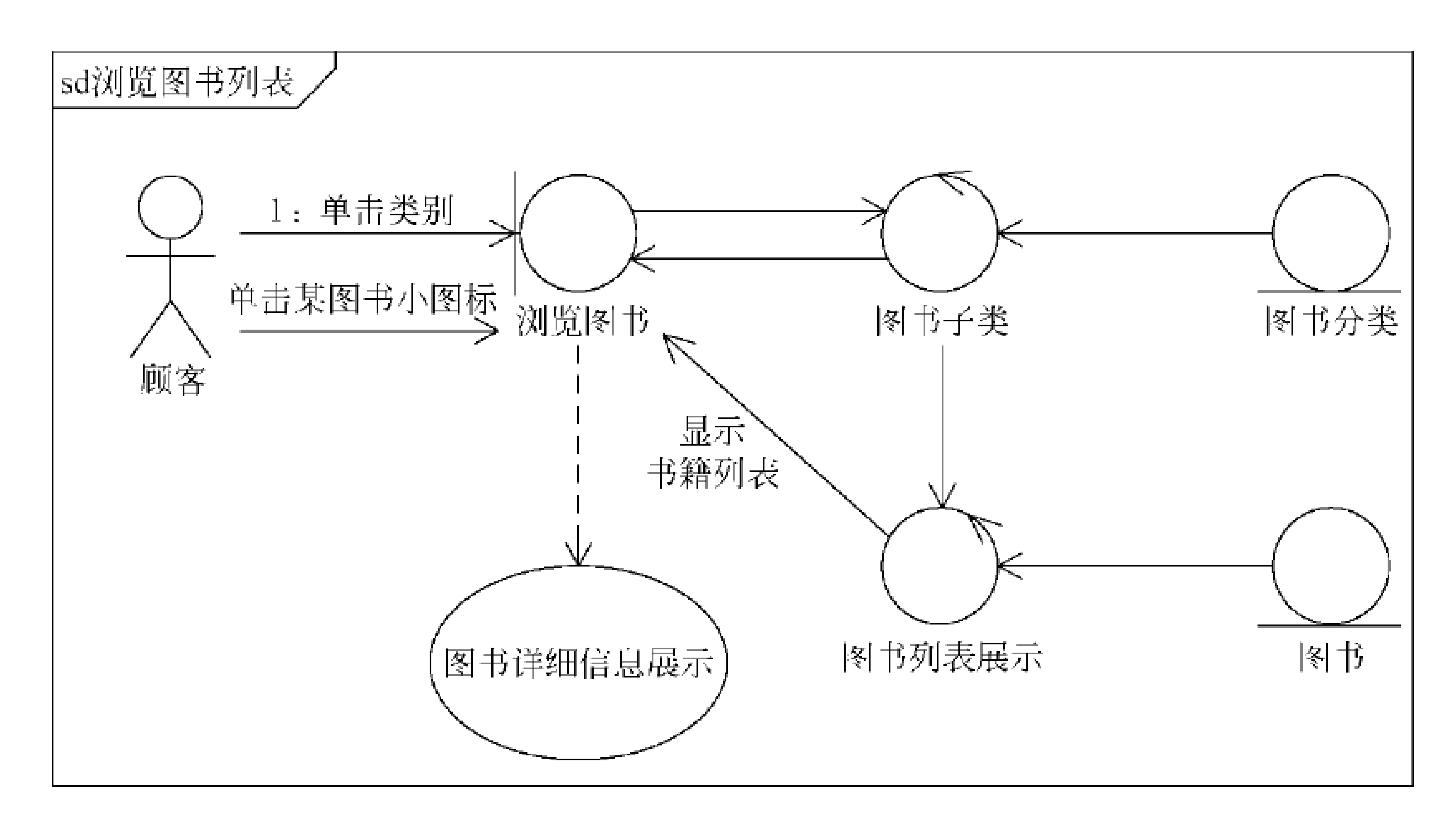


图 1-6　浏览图书列表交互页面

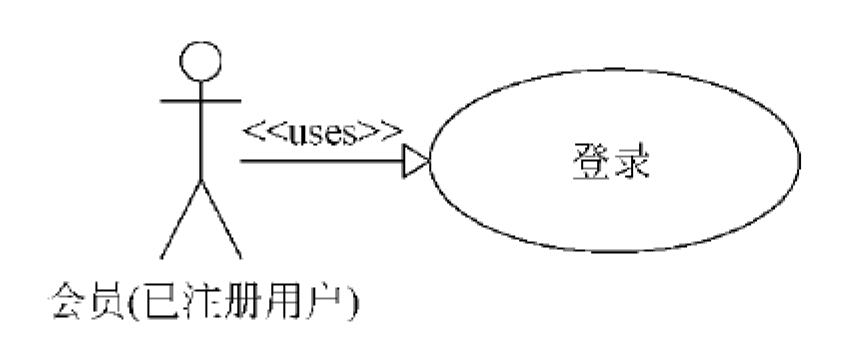


图 1-7　登录用例图

会员用户登录交互页面如图 1-8 所示。

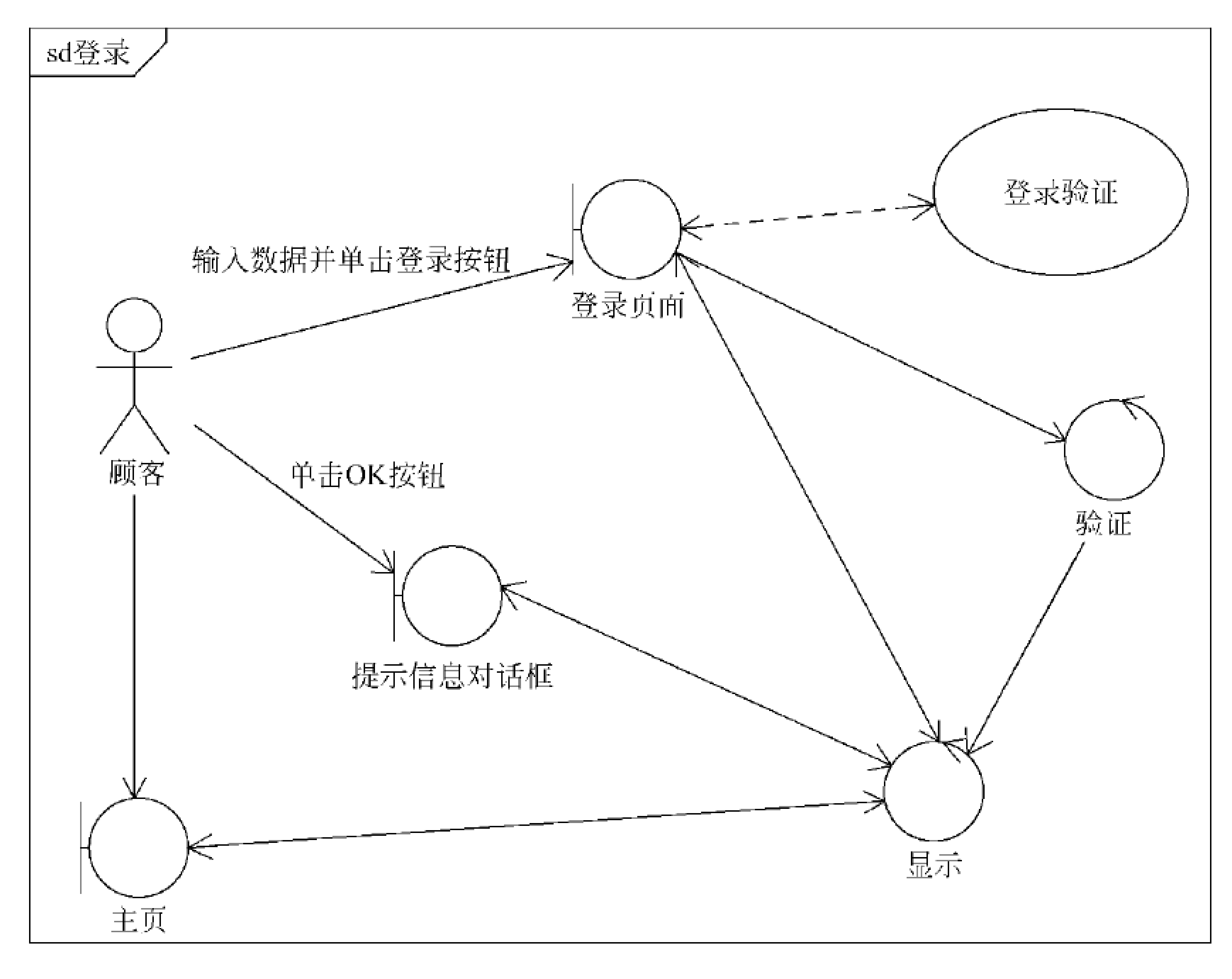


图 1-8　会员用户登录交互页面

登录用例描述如表 1-3 所示。

表 1-3　登录用例的细化描述

用例名称	登录
用例描述	会员登录系统
参与者	会员(已注册用户)
基本操作流程	① 用户单击主页中的“登录”链接。系统显示 LogIn. aspx 页面。用户输入其用户 ID 和密码,然后提交登录信息 ② 系统根据永久性账号数据对登录信息进行验证 ③ 登录成功,返回到主页
可选操作流程	如果用户单击 LogIn. aspx 页面上的“还没注册?”链接,系统将调用“注册”用例 如果用户输入的用户 ID 不正确,系统将显示一条消息,以指出这一点并提示用户输入其他的 ID 或点击“注册新账户”链接 如果用户输入的密码不正确,系统将显示一条消息,以指出这一点并提示用户重新输入密码

(5) 注册账户用例分析与描述。

注册账户用例图如图 1-9 所示。

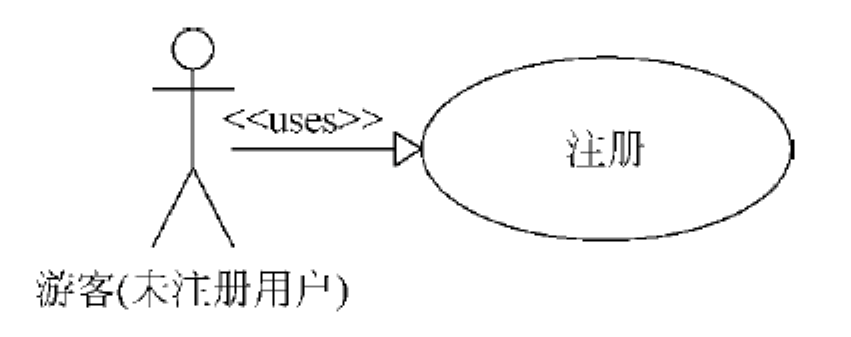


图 1-9　注册账户用例

注册账户用例描述如表 1-4 所示。

表 1-4　注册账户用例的细化描述

用例名称	注册
用例描述	游客注册成为会员
参与者	游客
基本操作流程	① 游客用户输入其 E-mail 地址、密码(两次),手机号码以及系统自动生成的图片验证码后提交注册信息 ② 系统确保游客用户提供的数据是有效的,然后使用这些数据进行保存(其中密码要求使用 MD5 加密形式保存)后,系统返回到首页,首页上的登录链接更换显示为“您好,用户的 E-mail 名”和一个“退出”链接
可选操作流程	如果游客用户没有提供 E-mail 或 E-mail 地址格式不正确,系统将显示一个错误消息提示 如果游客用户提供的密码太短,系统将显示一个错误消息提示(6～10 位) 如果游客用户提供的密码太简单,系统将显示一个错误消息提示 如果游客用户两次输入的密码不同,系统将显示一个错误消息提示 如果游客用户要创建的 E-mail 账号已经存在,系统将显示一个错误消息提示 如果游客用户输入的手机号格式不正确,系统将显示一个错误消息提示 如果游客用户输入的验证码,系统将显示一个错误消息提示

(6) 查询订单用例分析与描述。

查询订单用例图如图 1-10 所示。

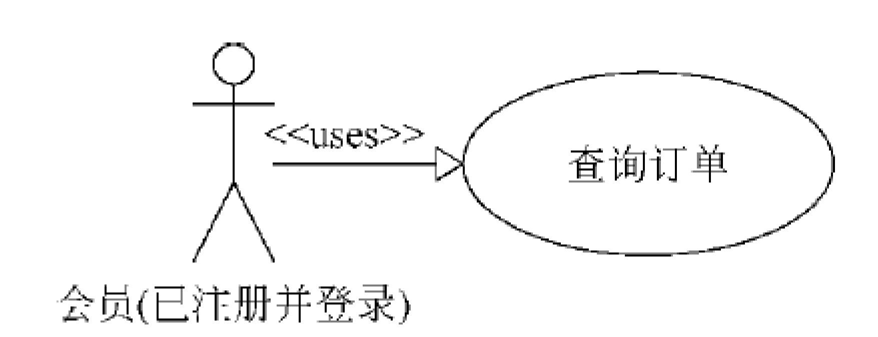


图 1-10 查询订单用例

查询订单用例描述如表 1-5 所示。

表 1-5 查询订单用例的细化描述

用例名称	查询订单
用例描述	查看历史订单信息
参与者	会员用户
基本操作流程	用户以列表形式查看历史订单信息，系统返回所有历史订单信息。 用户输入订单单号、订单状态、选择订单日期任意信息后，选择查询功能，系统根据查询信息内容返回数据库内容，查询订单信息
可选操作流程	用户输入错误的订单号码，系统返回空白订单信息 用户输入空白订单号码，系统提示用户输入订单号

对于其他用例，读者可以加以细化描述，不予全部列出。

任务 1-2-2 "新知书店"项目系统设计

【任务描述】

需求分析完成之后，得到的文档就是需求规格说明书，系统设计的任务就是对需求分析中的各功能模块描述、数据字典等进行物理实现，如公共类、每个功能模块实现(业务逻辑)、模块之间的接口、数据库、系统各部分界面、应用系统配置等的设计，形成的系统设计文档可以作为编码的依据。

当前大多数信息管理系统或动态网站其实都是对业务信息的管理，这些都离不开数据库。通过需求分析中的实体进行分析，画出实体联系(E-R)图，对关系型数据库进行设计，是系统设计中的重要组成部分。

【任务实施】

根据"新知书店"项目系统确定的边界范围，确定需求分析设计的功能模块，采用 Visio 或 Word 绘制总体结构图、系统技术框架图和前后台的实现流程图。在 ASP.NET 网站实现时，可以采用面向对象的设计方法(Object Oriented Design，OOD)和主流的三层架构设计模式，根据用例图、数据及业务流程图的描述，进行数据库表及视图的设计、表现层(User Interface，UI，即页面)设计、业务逻辑层(Business Logic Layer，BLL)、数据访问层(Data Access Layer，DAL)设计，绘制模块业务流程实现的顺序图或活动图。

1. 数据库的设计

根据前面的业务需求分析可知，"新知书店"系统主要对图书、用户及订单等对象进行有

效管理，实现图书信息浏览及管理、用户管理与在线购物等功能，通过需求分析可确定该系统涉及的实体有图书、用户、购物车、订单等。

可以用表格说明数据库表，表1-6～表1-14为“新知书店”项目重要数据库表的描述。在“新知书店”项目需求分析中，图书类别尽管只是图书的一个属性，但数据库设计中一般将其分离并创建成单独的表，这是一种面向对象编程(Object Oriented Programming，OOP)的思想，这样做有利于系统的扩展，便于使用。订单包括多种图书且数量不一，因此用两张表共同体现订单实体，在Orders表中存放订单的基本信息：订单编号、所属用户编号、下订单的时间及总价。OrderBook表中则存放订单的详细信息，每条记录体现某条订单记录所包含的某种图书的购买信息。

1）图书信息表Books

图书信息表用来记录图书的信息，其结构如表1-6所示。

表1-6　Books表结构

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	图书编号，自增
2	Title	nvarchar			否	图书名称
3	Author	nvarchar			否	作者姓名
4	PublisherId	int			否	出版社编号
5	PublishDate	datetime			否	出版日期
6	ISBN	nvarchar			否	图书出版号
7	UnitPrice	money			否	单价
8	ContentDescription	nvarchar			是	内容简介
9	TOC	nvarchar			是	目录
10	CategoryId	int			否	图书分类编号
11	Clicks	int			否	点击数

2）图书分类表Categories

图书分类表用来存储图书类别信息，其结构如表1-7所示。

表1-7　Categories表结构

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	图书分类编号
2	Name	nvarchar			否	分类名称
3	PId	int			是	父类编号
4	SortNum	int			是	排序号

3）订单表Orders

订单表用于存储每笔订单的基本信息，其结构如表1-8所示。

4）订单详细信息表OrderBook

订单详细信息表用于存储订单的详细信息，其结构如表1-9所示。

5）出版社信息表Publishers

出版社信息表用于存储出版社基本信息，其结构如表1-10所示。

表 1-8 Orders 表结构

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	订单编号
2	OrderDate	datetime			否	订购日期
3	UserId	int			否	所属用户编号
4	TotalPrice	decimal			否	总金额
5	State	int			否	状态

表 1-9 OrderBook 表结构

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是		否	编号,自动增量
2	OrderID	int			否	订单编号
3	BookID	int			否	图书编号
4	Quantity	int			否	数量
5	UnitPrice	money			否	单价

表 1-10 Publishers 表结构

序号	字段名	数据类型	主键	外键	允许空	备注
1	Id	int	是	是	否	出版社编号
2	Name	nvarchar			否	出版社名称

6）购物车信息表 TemporaryCart

购物车信息表用于存储用户购物的信息,其结构如表 1-11 所示。

表 1-11 TemporaryCart 表结构

序号	字段名	数据类型	主键	外键	允许空	备注
1	Id	int	是		否	购物车编号
2	CreateTime	datetime			否	创建时间
3	BookId	int			否	图书编号
4	UserId	int			否	所属用户编号

7）用户角色表 UserRoles

用户角色表用于存储系统用户角色信息,其结构如表 1-12 所示。

表 1-12 UserRoles 表结构

序号	字段名	数据类型	主键	外键	允许空	备注
1	Id	int	是	是	否	角色 Id
2	Name	nvarchar			否	角色名称

8）用户状态表 UserStates

用户状态表用于存储系统用户状态信息,其结构如表 1-13 所示。

表 1-13 UserStates 表结构

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	状态编号
2	Name	nvarchar			否	状态名称

9）用户基本信息表 Users

用户信息表用于存储用户的基本信息，其结构如表 1-14 所示。

表 1-14 Users（用户信息表）

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	用户编号
2	LoginId	nvarchar			否	登录名
3	LoginPwd	nvarchar			否	登录密码
4	Name	nvarchar			否	用户真实姓名
5	Address	nvarchar			否	地址
6	Phone	nvarchar			否	电话
7	Mail	nvarchar			是	电子邮箱
8	UserRoleId	int			否	角色编号
9	UserStateId	int			否	状态编号

2. 表现层设计

表现层（UI）设计主要运用 HTML 或 ASP.NET 来设计，表现层重点是页面的设计，一方面表现层是用户访问“新知书店”的窗口，另一方面也是管理员操作结果的展示，在设计时要求能满足功能需求，方便用户，美观大方，图 1-11 是用户管理模板中管理员查询用户信息的界面。

您现在的位置： 新知书店 > 管理员后台 > 用户管理 > 管理用户

用户名	姓名	地址	电话	Email	操作	
bobo	张三	北京	13765277988	bobo@163.com	选择	删除
admin	admin	admin	13456	admin@163.com	选择	删除
恰嬉猫	qiaximao	上海市华夏路100号	13774210000	qxm@163.com	选择	删除

1 2 3 4

图 1-11 管理员查询用户信息界面

3. 业务逻辑层设计

负责业务处理和数据传递，它包含了与核心业务相关的逻辑，实现业务规则和业务逻辑。业务逻辑层（BLL）处于数据访问层与表示层之间，还作为表示层和数据访问层的桥梁，实现数据的传递和处理，起到了数据交换中承上启下的作用，对于数据访问层而言，它是调

用者；对于表示层而言，它却是被调用者。图 1-12 为用户注册流程图，图 1-13 为用户管理模块的用户注册业务逻辑层类图。

4. 数据访问层设计

数据访问层(DAL)封装了操作数据表的方法，设计时首先将数据库连接类单独设计或将数据库连接字符串写在配置文件 web.config 中，在具体模块的数据访问层类调用。用户管理模块的用户注册数据访问层类图如图 1-14 所示。

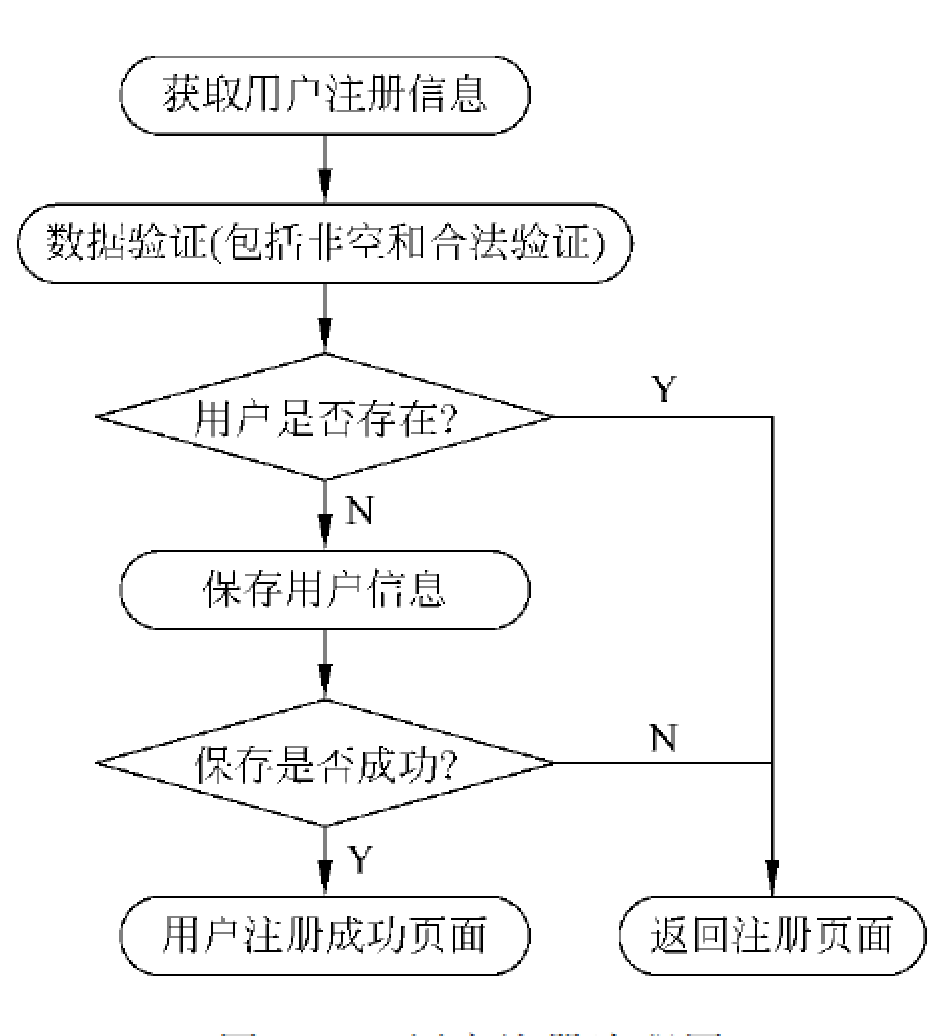


图 1-12　用户注册流程图



图 1-13　用户管理业务逻辑层类图



图 1-14　用户注册数据访问层类图

这里读者需要读者注意的是本任务不是整个“新知书店”项目模块的设计，而是项目中各模块的设计思路与步骤，具体到本任务，采用用户管理模块的用户注册为例进行介绍，其他模块的详细设计将在后续单元任务中介绍。

1.3 单元实训

博客系统是目前非常流行的网络信息发布方式，用户群体大，受到很多人的青睐。本书的单元实训任务就是构建一个属于自己的博客系统。由于博客系统的功能庞大，为了让读者在有限的时间内练习最重要的知识点，我们要完成的博客系统所确定的业务需求比较简单。博客提供了一个注册会员的功能，会员可以登录博客系统、发表文章、编辑文章、发表评论、查看文章；没有注册的会员，也就是匿名用户也可以查看文章、评论文章(匿名评论)、查看评论等。本单元读者需完成“博客系统”的数据库设计。

设计“博客系统”数据库

【需求说明】

- 用户应包含的内容为：用户编号、登录名、登录密码、姓名、QQ、E-Mail 等。

- 文章应包含的内容为：文章编号、文章所属用户编号、文章标题、内容、创建时间、点击次数等。
- 文章评论应包含的内容为：评论编号、评论的文章编号、评论人、评论时间、评论内容等。

读者根据上述描述创建数据库 MyBlog 及其三张表并建立关系，表结构如表 1-15～表 1-17 所示。

表 1-15　Users(用户信息表)

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	用户编号
2	LoginId	nvarchar			否	登录名
3	LoginPwd	nvarchar			否	登录密码
4	Name	nvarchar			否	用户真实姓名
5	QQ	nvarchar			否	QQ 号
6	Mail	nvarchar			是	电子邮箱

表 1-16　Articles(文章信息表)

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是	是	否	文章编号
2	AuthorId	int			否	文章所属用户编号
3	Title	int			否	文章标题
4	Contents	text			否	文章内容
5	PubDate	datetime			否	发表日期
6	Clicks	int			是	点击次数

表 1-17　Comments(文章评论信息表)

序号	字段名	数据类型	主键	外键	允许空	说明
1	Id	int	是		否	评论编号
2	ArticleId	int			否	评论所属文章编号
3	AuthorName	nvarchar			否	评论人姓名
4	Contents	text			否	评论内容
5	PubDate	datetime			否	评论日期

1.4　单元小结

本单元对“新知书店”整个系统的需求分析和软件设计进行了简单的描述，具体到每个模块或子系统，将在后续个单元任务中详细介绍，本单元只作了全局设计和解释了要完成的任务。对于没有太多开发经验和业务系统知识的读者而言，最好的办法就是要参考已有的

类似软件系统、开发文档、文档模板，然后按照用户的要求进行分析和设计，这样才能快速地实现需求分析和软件设计。本单元所阐述的知识体系如图 1-15 所示。

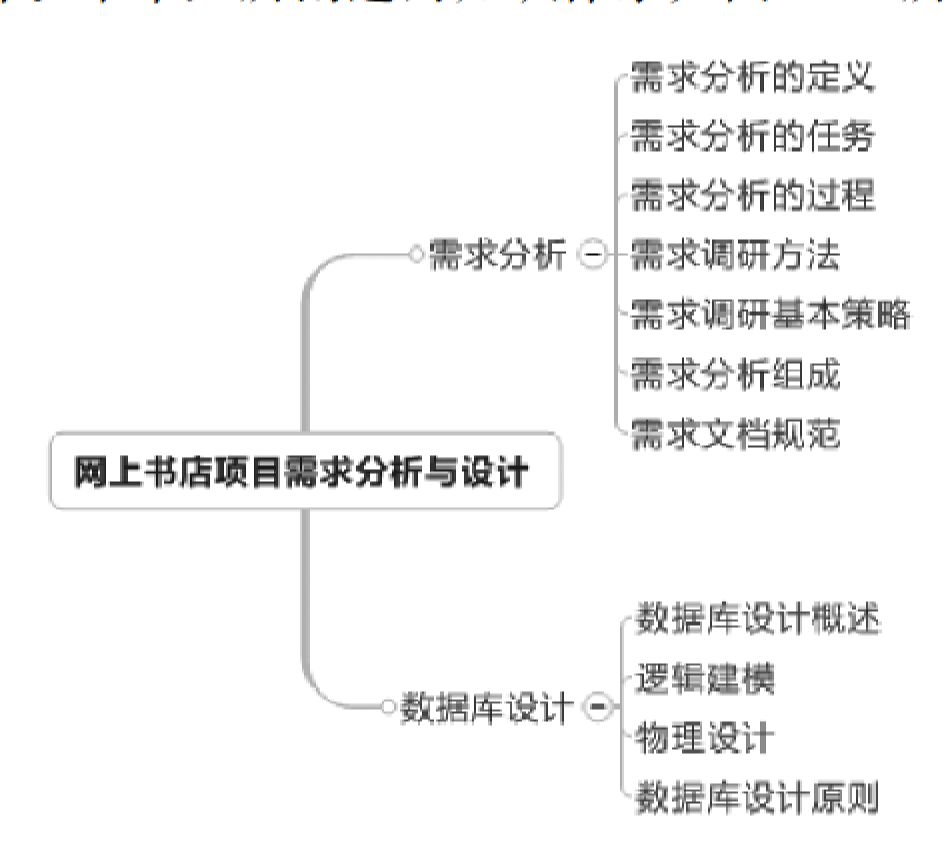


图 1-15　项目需求分析与设计知识体系

单元2 ASP.NET基础及开发环境构建

教学目标：

- 会安装与配置 ASP.NET 网站的运行环境。
- 会搭建 ASP.NET 网站的开发环境。
- 会创建简单的 Web 网站。
- 了解 ASP.NET 文档的结构。
- 了解 ASP.NET 的运行机制和 ASP.NET 的文件类型。
- 理解静态网页与动态网页的概念及其工作原理。

2.1 知识准备

2.1.1 .NET Framework 概述

.NET Framework 是微软近年来主推的应用程序开发框架，是一套语言独立的应用程序开发框架。微软公司发布 .NET Framework 的目的是使开发人员可以更容易地建立网络应用程序和网络服务，.NET Framework 以及针对设备的 .NET Framework 简化版为 XML Web 服务和其他应用程序提供了一个高效安全的开发环境，并全面支持 XML。.NET Framework 提供跨平台和跨语言的特性，使用 .NET 框架，配合微软公司的 Visual Studio 集成开发环境，可大大提高程序员的开发效率，甚至初学者也能够快速构建功能强大、实用、安全的网络应用程序。有的功能甚至不需要任何开发代码，经过简单的操作就可以实现。

1. .NET Framework 的概念

.NET Framework 是一个开发和运行环境，它使得不同的编程语言(如 C# 和 VB.NET 等)和运行库能够无缝地协同工作，简化开发和部署各种网络集成应用程序或独立应用程序，如 Windows 窗体应用程序、ASP.NET Web 应用程序、WPF 应用程序、移动应用程序或 Office 应用程序。.NET Framework 的基本结构如图 2-1 所示。

1) 公共语言运行库

公共语言运行库(Common Language Runtime，CLR)，又称为公共语言运行环境，是 .NET Framework 的基础。运行库作为执行时管理代码的代理，提供了内存管理、线程管理

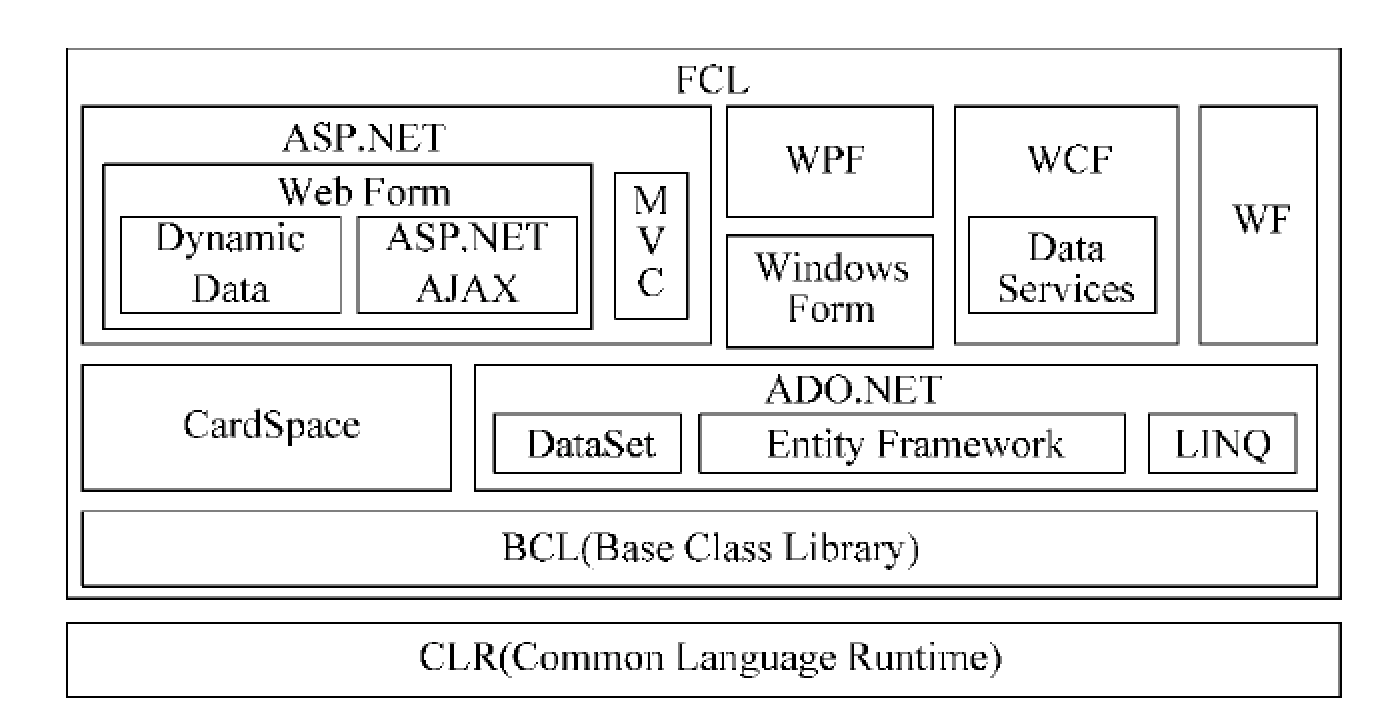


图 2-1 .NET Framework 的基本结构

和远程处理等核心服务，并且还强制实施严格的类型安全检查，以提高代码准确性。

在运行库的控制下执行的代码称作托管代码。托管代码使用基于公共语言运行库的语言编译器开发生成，具有许多优点：跨语言集成、跨语言异常处理、增强的安全性、版本控制和部署支持、简化的组件交互模型、调试和分析服务等。

在运行库之外运行的代码称作非托管代码。COM 组件、ActiveX 接口和 Win32 API 函数都是非托管代码的示例。使用非托管代码方式可以提供最大限度的编程灵活性，但不具备托管代码方式所提供的管理功能。

2）.NET Framework 类库

.NET Framework 类库（.NET Framework Class Library，FCL）是一个与公共语言运行库紧密集成、综合性的、面向对象的类型集合。使用该类库，可以高效率开发各种应用程序，包括控制台应用程序、Windows GUI 应用程序（Windows 窗体）、ASP.NET Web 应用程序、XML Web Services、Windows 服务等。

.NET Framework 类库包括类、接口和值类型。类库提供对系统功能的访问，以加速和优化开发过程。.NET Framework 类型符合公共语言规范（Common Language Specification，CLS），因而可在任何符合 CLS 的编程语言中使用，实现各语言之间的交互操作。

.NET Framework 类库由基础类库（Base Class Library，BCL）和各种应用程序框架类库组成。基础类库主要提供下列功能：

- 表示基础数据类型和异常。
- 封装数据结构。
- 执行 I/O。
- 访问关于加载类型的信息。
- 调用.NET Framework 安全检查。

各种应用程序框架类库提供构建相应应用程序的功能：

- 数据访问（ADO.NET）。
- Windows 窗体（Windows Form）。
- Web 窗体（ASP.NET）。

2. .NET Framework 的功能特点

.NET Framework 提供了基于 Windows 的应用程序所需的基本架构，开发人员可以基于.NET Framework 快速建立各种应用程序解决方案。.NET Framework 具有下列功能特点。

1）支持各种标准互联网协议和规范

.NET Framework 使用标准的 Internet 协议和规范（如 TCP/IP、SOAP、XML 和 HTTP 等），支持实现信息、人员、系统和设备互连的应用程序解决方案。

2）支持不同的编程语言

.NET Framework 支持多种不同的编程语言，因此开发人员可以选择他们所需的语言。公共语言运行库提供内置的语言互操作性支持，公共语言运行库通过指定和强制公共类型系统以及提供元数据为语言互操作性提供必要的基础。

3）支持用不同语言开发的编程库

.NET Framework 提供了一致的编程模型，可使用预打包的功能单元（库），从而能够更快、更方便、更低成本地开发应用程序。

4）支持不同的平台

.NET Framework 可用于各种 Windows 平台，从而允许使用不同计算平台的人员、系统和设备联网，例如，使用 Windows XP/Vista/7 等台式机平台或 Windows CE 之类的设备平台的人员可以连接到使用 Windows Server 2003/2008 的服务器系统。

3. .NET Framework 环境

操作系统/硬件、公共语言运行库、类库以及应用程序（托管应用程序、托管 Web 应用程序、非托管应用程序）之间的关系如图 2-2 所示。

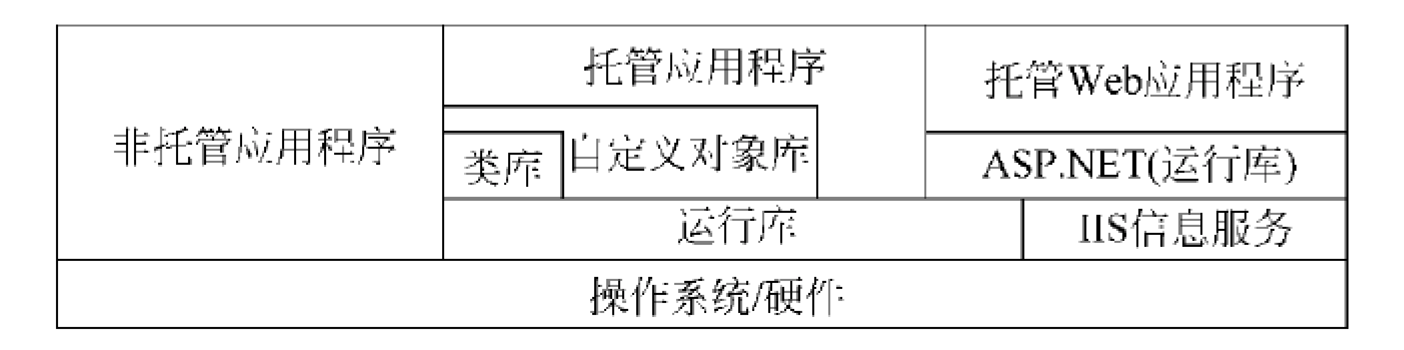


图 2-2　.NET Framework 环境

4. .NET Framework 的主要版本

目前，.NET Framework 主要包含下列版本：1.0、1.1、2.0、3.0、3.5、4.0，支持带最新 Service Pack 的桌面 Windows 操作系统。与之相对应，.NET Compact Framework 可用作所有 Microsoft 智能设备（包括 Pocket PC 设备、Pocket PC Phone Edition、Smartphone 设备以及其他安装有 Windows Embedded CE 的设备）中的操作系统组件。

其中，1.0、1.1、2.0 和 4.0 版是彼此完全独立的，即对于其中任何一个版本都可以独立存在于某计算机上，无论计算机上是否存在其他版本。当 1.0、1.1 和 2.0 版位于同一台计算机上时，每个版本都有自己的公共语言运行库、类库和编译器等。应用程序开发人员可以选择面向特定的版本开发和部署应用程序。各版本之间的关系如图 2-3 所示。

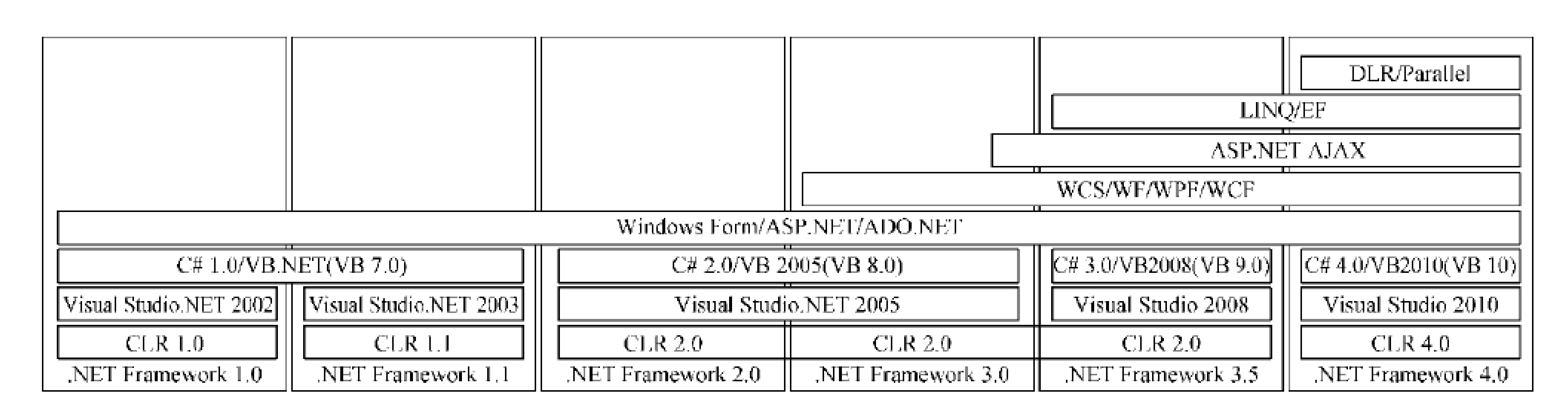


图 2-3 .NET Framework 各版本之间的关系

2.1.2 Web 基础知识

1. HTTP 协议

HTTP(Hyper Text Transfer Protocol)协议,即超文本传输协议,是在 Internet 中进行信息传送的协议,浏览器默认使用这个协议。

从浏览器向 Web 服务器发出的搜索某个 Web 网页的请求叫作 HTTP 请求。Web 服务器收到 HTTP 请求后,就会按照请求的要求,寻找相应的网页。如果找到,就把网页的 HTML(Hypertext Markup Language,超文本标记语言)代码通过 Internet 传回浏览器;如果没有找到,就发送一个错误信息给发出 HTTP 请求的浏览器,后面的这些操作就叫作 HTTP 响应。

HTTP 协议是一个无状态协议,也就是说,使用该协议时,不同的请求之间不会保存任何信息。每个请求都是独立的,它不知道现在的请求是第一次发出还是第二次或是第三次发出,也不知道这个请求的发送来源。当用户请求到所要的网页后,就会断开与 Web 服务器的链接。

2. Web 服务器和浏览器

Web 服务器就是安装了 Web 服务器软件的计算机,它可以为提出 HTTP 请求的浏览器提供 HTTP 响应。比较常见的 Web 服务器软件有 Apache 和 IIS。

浏览器是运行在客户机上的程序,用户可以用它来浏览服务器中的可用资源,因此称为浏览器。当客户进行网页浏览时,由客户的浏览器执行来自服务器的 HTML 代码,并将其内容显示给客户。

3. C/S 模式与 B/S 模式

C/S 和 B/S 是目前开发模式技术架构的两大主流技术。C/S 模式最早是由美国 Borland 公司研发,而 B/S 模式是由美国微软公司研发的。

1) C/S 模式

C/S(Client/Server,客户机/服务器)模式是一种软件系统体系结构。

2) B/S 模式

B/S(Browser/Server,浏览器/服务器)模式是随着 Internet 技术的兴起,对 C/S 模式的一种变化或改进。在这种模式下,用户工作界面是通过 Web 浏览器来实现的。B/S 模式的

最大好处是能够实现不同人员从不同地点以不同的接入方式访问和操作共同的数据，这大大减轻了系统维护与升级的成本和工作量，降低了用户的总体成本；最大的缺点是对外网依赖性太强。

4. Web的访问原理

Web应用程序是基于B/S结构的。下面首先介绍客户端和服务器端的概念，然后详述静态网页和动态网页的工作原理。

1）客户端和服务器端

一般来说，凡是提供服务的一方称为服务器端，而接受服务的一方称为客户端。例如，当用户浏览搜狐主页的时候，搜狐网站所在的服务器就称为服务器端，而用户自己的计算机就称为客户端，如图2-4所示。

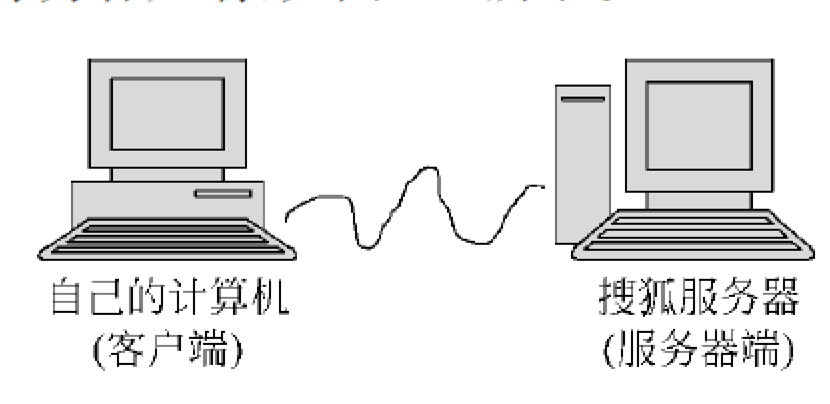


图2-4　客户端和服务器端示例图

如果在自己的计算机上安装了Web服务器软件，其他浏览者通过网络就可以访问该计算机，那么它就是服务器端。在调试程序时，往往把自己的计算机既作为服务器端，又作为客户端。

2）静态网页的工作原理

静态网页也称为普通网页，是相对动态网页而言的。静态并不是指网页中的元素都是静止不动的，而是指网页文件里没有程序代码，只有HTML（超文本标记语言）标记，一般后缀为.htm、.html、.shtml或.xml等。静态网页中可以包括GIF动画，鼠标经过Flash按钮时，按钮可能会发生变化。静态网页一经制成，内容就不会再变化，不管何人何时访问，显示的都是一样的内容。如果要修改网页的内容，就必须修改其源代码，然后重新上传到服务器。

对于静态网页，用户可以直接双击打开，看到的效果与访问服务器是相同的。这是因为在用户访问该页面之前，网页的内容就已经确定，无论用户何时访问，以怎样的方式访问，网页的内容都不会再改变。静态网页工作流程可以分为以下4个步骤：

（1）编写一个静态网页文件，并在Web服务器上发布。

（2）用户在浏览器的地址栏中输入此静态网页文件的URL（统一资源定位符）并按回车，浏览器发送访问请求到Web服务器。

（3）Web服务器找到此静态网页文件的位置，并将它转换为Html流传送到用户的浏览器。

（4）浏览器收到HTML流，显示此网页的内容。

在步骤(2)～(4)中，静态网页的内容不会发生任何变化，其原理如图2-5所示。

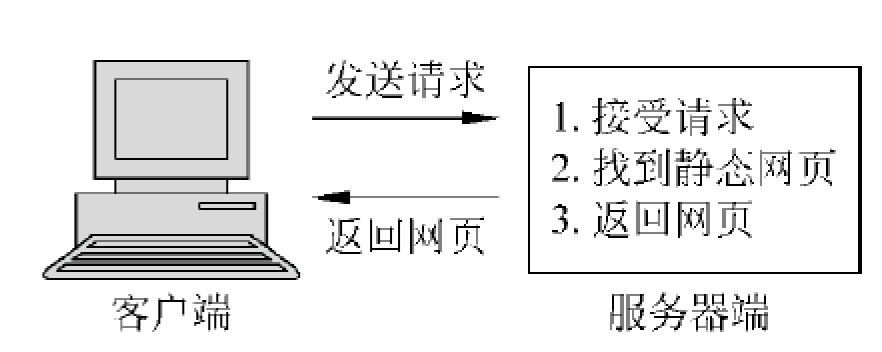


图2-5　静态网页工作原理图

3）动态网页的工作原理

动态网页是指在网页文件中除了HTML标记外，还包括一些实现特定功能的程序代码，这些程序代码使得浏览器与服务器之间可以发生交互，即服务器端可以根据客户端的不同请求动态产生网页内容。动态网页的后缀通常根据所用的程序设计语言的不同而不同，

一般为.asp、.aspx、.cgi、.php、.perl、.jsp 等。动态网页可以根据不同的时间、不同的浏览者而显示不同的信息。常见的留言板、论坛、聊天室都是用动态网页实现的。

动态网页的工作相对复杂，不能直接双击打开，动态网页的工作流程分为以下 4 个步骤：

(1) 编写一个动态网页文件，其中包括程序代码，并在 Web 服务器上发布。

(2) 用户在浏览器的地址栏中输入该动态网页文件的 URL(统一资源定位符)并按回车，浏览器发送访问请求到 Web 服务器。

(3) Web 服务器找到此动态网页文件的位置，并根据其中的程序代码动态创建 HTML 流传送到用户的浏览器。

(4) 浏览器收到 HTML 流，显示此网页的内容。

从整个工作流程中可以看出，用户浏览动态网页时，需要在服务器上动态执行该网页文件，将含有程序代码的动态网页转化为标准的静态网页，最后把生成的静态网页发送给用户，其工作原理如图 2-6 所示。

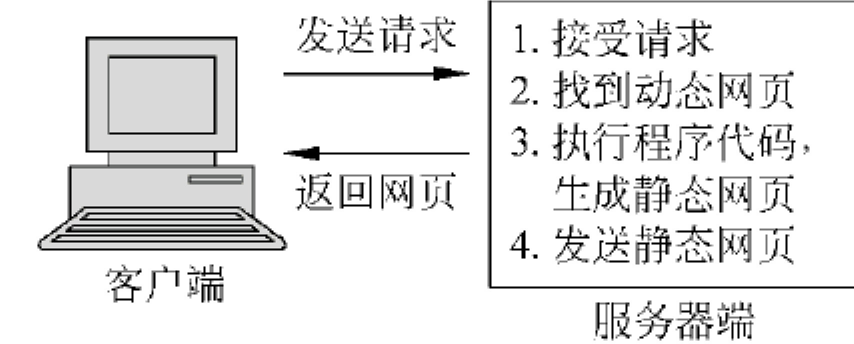


图 2-6 动态网页工作原理图

2.1.3 ASP.NET 简介

ASP.NET 是 Microsoft.NET 的一部分，是 Active Server Page(简称 ASP)的另一个版本，是建立在微软新一代.NET 平台架构上、建立在公共语言运行库上，在服务器后端为用户提供强大的企业级 Web 应用服务的编程框架。ASP.NET 提供了一种新的编程模型和结构，可生成伸缩性和稳定性更好的应用程序，并提供更好的安全保护。

ASP.NET 是一个已编译的、基于.NET 的环境，可以用任何与.NET 兼容的语言(包括 Visual Basic.NET、C# 和 JScript .NET)创作应用程序。另外，任何 ASP.NET 应用程序都可以使用整个.NET Framework。开发人员可以方便地获得这些技术的优点，其中包括托管的公共语言运行库环境、类型安全和继承等。

Microsoft 公司为 ASP.NET 设计了功能强大的代码、代码易于重用和共享，可用编译类语言编写策略，从而使程序员更易开发 Web 应用程序，满足不同客户的需求。

1. ASP.NET 的历史

1996 年，Microsoft 公司推出了 ASP(Active Server Page)1.0 版。它允许采用 VBScript/JavaScript 这些简单的脚本语言编写代码，允许将代码直接嵌入 HTML，从而使得设计动态 Web 页面的工作变得简单。在进行程序设计时，ASP 能够通过内置的组件，实现强大的功能(如 Cookie)。ASP 最显著的贡献就是推出了 ActiveX Data Objects(ADO)，它使得程序对数据库的操作变得十分简单。

1998 年，微软发布了 ASP 2.0 和 IIS 4.0。与前版相比，2.0 版最大的改进是外部的组件需要初始化。用户能够利用 ASP 2.0 和 IIS 4.0 建立各种 ASP 应用，而且每个组件有了自己单独的内存空间，可以进行事务处理。

2002 年推出的新一代体系结构——Microsoft.NET 的一部分，用来在服务器端构建功能强大的 Web 应用，包括 Web 窗体(Web Form)和 Web 服务(Web Services)两部分。

2003 年，Microsoft 公司发布了 Visual Studio.NET 2003(简称 VS 2003)，提供了在 Windows 操作系统下开发各类基于.NET 框架的全新的应用程序开发平台。

2005 年，.NET 框架从 1.0 版升级到 2.0 版，Microsoft 公司发布了 Visual Studio.NET 2005(简称 VS 2005)。相应的 ASP.NET 1.0 也从得到了升级，成为 ASP.NET 2.0。它修正了以前版本中的一些错误(Bug)并在移动应用程序开发，代码安全以及对 Oracle 数据库和 ODBC 的支持等方面都做了很多改进。

2008 年，Visual Studio.NET 2008(简称 VS 2008)问世了，ASP.NET 相应的从 2.0 版升级到 3.5 版。

2010 年，Microsoft 公司发布 Visual Studio.NET 2010 正式版本。

2. ASP.NET 的优点

ASP.NET 是 Microsoft .NET Framework 的一部分，是一种可以在高度分布的 Internet 环境中简化应用程序开发的环境。.NET Framework 包含公共语言运行库，它提供了各种核心服务，如内存管理、线程管理和代码安全，同时也包含.NET Framework 类库。.NET Framework 是一个开发人员用于创建应用程序的综合的、面向对象的类型集合。

ASP.NET 的优点主要表现在以下几个方面。

1) 可管理性

ASP.NET 使用基于文本的、分级的配置系统，简化了将设置应用于服务器环境和 Web 应用程序的工作。因为配置信息是被存储为纯文本格式的，因此可以在没有本地管理工具的帮助下应用新的设置。

注意：配置文件的任何变化都可以被自动检测到并应用于应用程序。有关这方面的详细信息，请参阅 ASP.NET 配置相关知识。

2) 安全性高

ASP.NET 为 Web 应用程序提供了默认的授权和身份验证方案。开发人员可以根据应用程序的需要很容易地添加、删除或替换这些方案。

3) 易于部署

ASP.NET 应用程序可以部署到服务器上，并且不需要重新启动服务器，甚至在部署或替换运行的已编译代码时也不需要重新启动。

4) 增强的性能

ASP.NET 是运行在服务器上的已编译代码。与传统的 ASP 不同，ASP.NET 能利用早期绑定、实时(Just In Time，JIT)编译、本机优化和全新的缓存服务来提高性能。

5) 灵活的输出缓存

根据应用程序的需要，ASP.NET 可以缓存页数据、页的一部分或整个页。缓存的项目可以依赖于缓存中的文件或其他项目，或者可以根据过期策略进行刷新。

6) 移动设备支持

ASP.NET 支持任何设备上的任何浏览器。开发人员使用与传统的桌面浏览器相同的编程技术，来处理新的移动设备。

7）扩展性和可用性

ASP.NET 具有特别专有的功能来提高群集的、多处理器环境的性能。此外，Internet 信息服务（Internet Information Server，IIS）和 ASP.NET 运行时密切监视和管理进程，以便在一个进程出现异常时，可在该位置创建新的进程使应用程序继续处理请求。

8）跟踪和调试

ASP.NET 提供了跟踪服务，该服务可在应用程序级别和页面级别调试过程中启用。可以选择查看页面的信息，或者使用应用程序级别的跟踪查看工具查看信息。在开发或应用程序处于生产状态时，ASP.NET 支持使用.NET Framework 调试工具进行本地和远程调试。当应用程序处于生产状态时，跟踪语句能够留在产品代码中而不会影响性能。

9）与.NET Framework 集成

ASP.NET 是.NET Framework 的一部分，整个平台的功能和灵活性对 Web 应用程序都是可用的，因此可从 Web 上流畅地访问.NET 类库及消息和数据访问解决方案。ASP.NET 是独立于语言之外的，所以开发人员能选择最适合应用程序的语言。另外，公共语言运行库的互用性还保存了基于 COM 开发的现有投资。

10）与现有 ASP 应用程序的兼容性

ASP 和 ASP.NET 可并行运行在 IIS Web 服务器上而互不冲突；不会发生因安装 ASP.NET 而导致现有 ASP 应用程序崩溃的可能。

注意： ASP.NET 仅处理具有.aspx 文件扩展名的文件，具有.asp 文件扩展名的文件继续由 ASP 引擎来处理。会话状态和应用程序状态并不在 ASP 和 ASP.NET 页面之间共享。

2.2 单元任务

任务 2-2-1 安装和配置 IIS Web 服务器

【任务描述】

在 Windows XP 操作系统中正确安装与配置 IIS 管理器，搭建 ASP.NET Web 应用程序的运行环境。

【任务实施】

(1) 选择"开始"→"控制面板"→"添加或删除程序"命令，显示如图 2-7 所示的"添加或删除程序"窗口，该窗口显示当前已经安装的程序。

(2) 在窗口的左侧选择"添加/删除 Windows 组件"图标，弹出如图 2-8 所示的"Windows 组件向导"对话框，找到"Internet 信息服务(IIS)"复选框，如果尚未安装，则其左侧的复选框不会被选中；如果复选框是不可选状态，则说明 IIS 的组件没有全部安装。否则说明 IIS 已经全部安装。

(3) 如果复选框没有被选中，则选中该复选框。如果复选框是不可选状态，则选中该项，单击"详细信息"按钮，弹出如图 2-9 所示的"Internet 信息服务器(IIS)"对话框。

(4) 选择要安装的选项。对于本书来说，一定要选中"公用文件"复选框。选择要安装

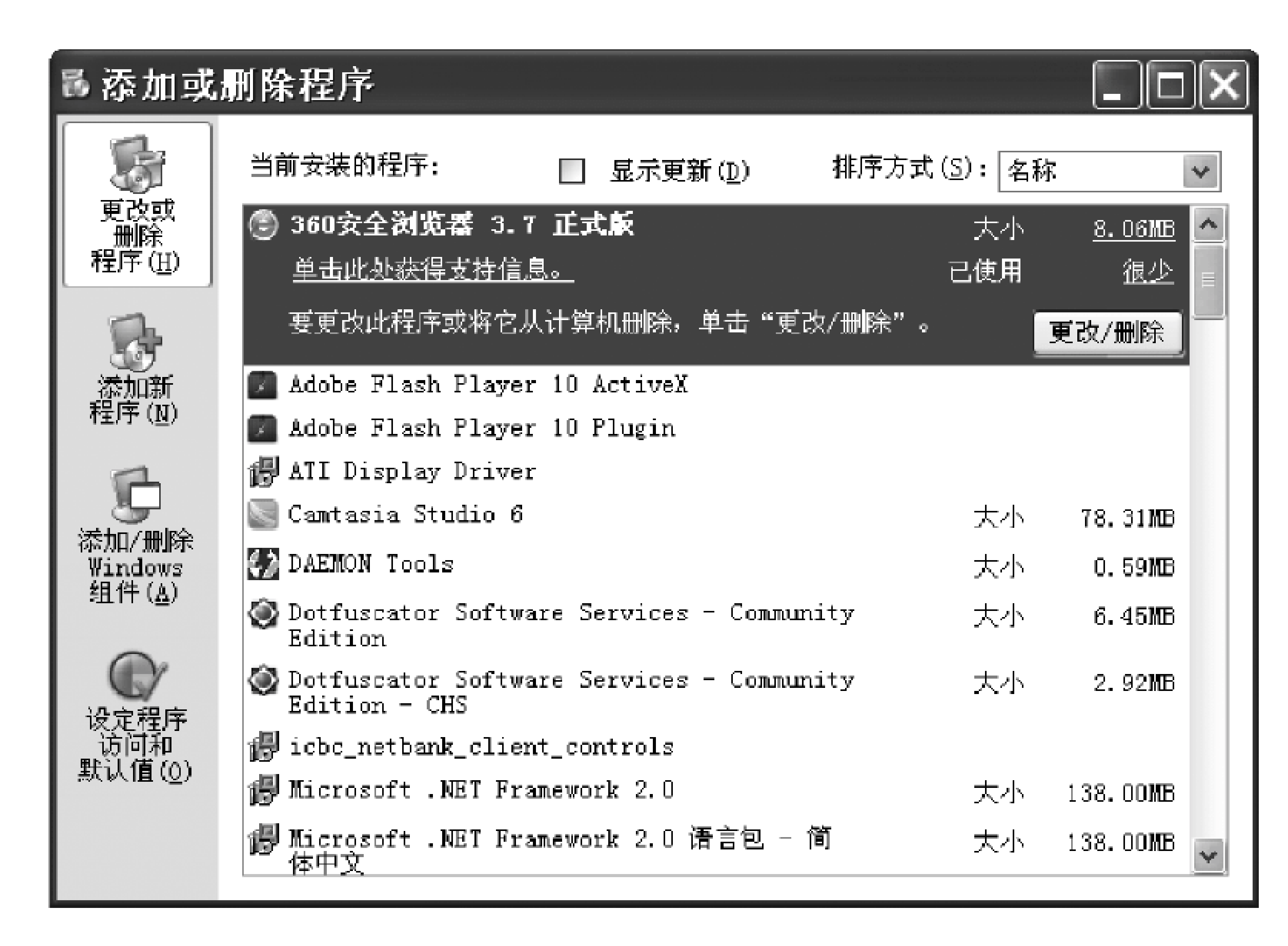


图 2-7　“添加或删除程序”窗口



图 2-8　“Windows 组件向导”对话框

的选项后，单击“确定”按钮，返回到“Windows 组件向导”对话框。单击“下一步”按钮安装 IIS，此时会提示用户将 Windows XP 系统盘放入光驱或者选择要安装的 IIS 文件。

(5) 放入 Windows XP 系统盘或选择要安装 IIS 文件后，根据提示依次单击“下一步”按钮，最后，单击“完成”按钮，完成 IIS 的安装。

(6) 选择“开始”→“控制面板”→“管理工具”→“Internet 信息服务”命令，弹出如图 2-10 所示的“Internet 信息服务”窗口，依次展开“根”节点、“网站”节点、“默认网站”节点。

(7) 右击“默认网站”节点，弹出如图 2-11 所示的菜单。可以选择“停止”命令关闭 IIS 服务，也可以选择“暂停”命令暂停 IIS 服务。

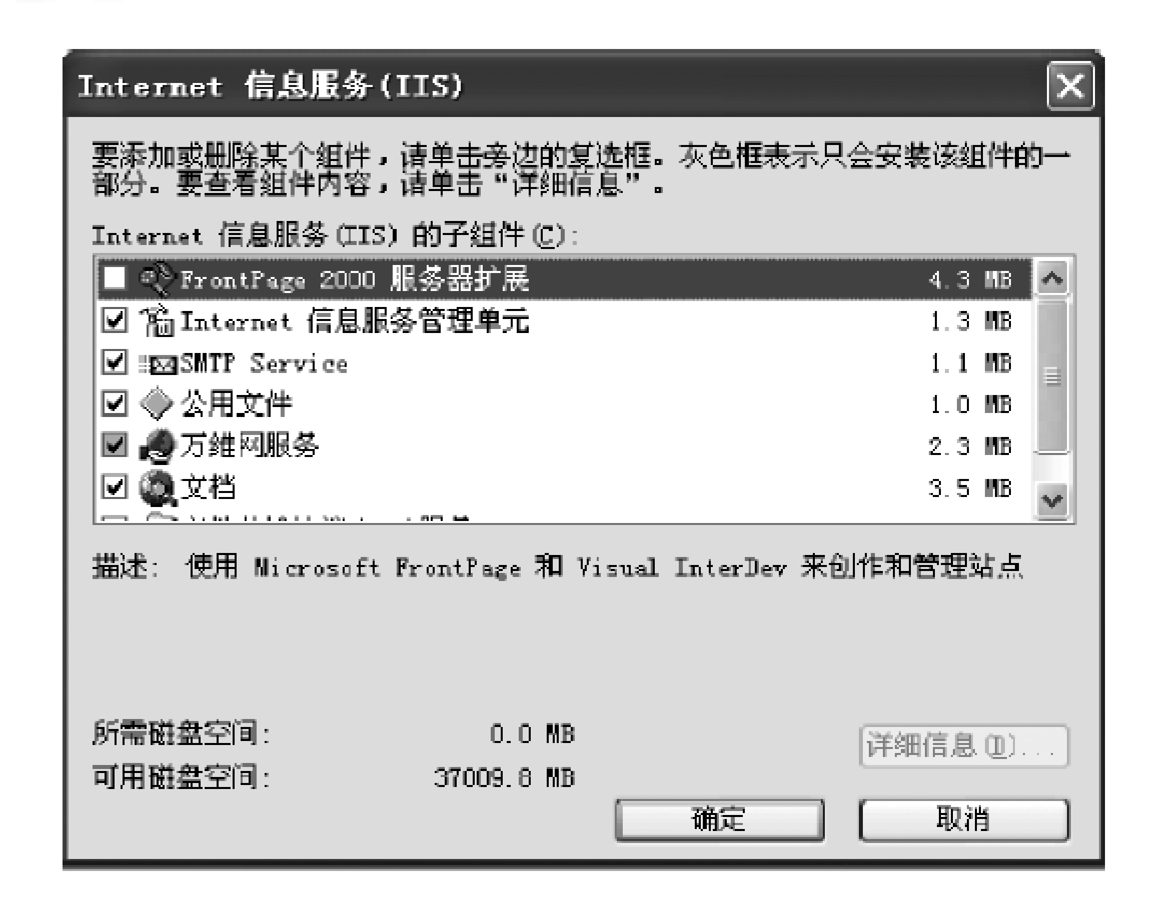


图 2-9 “Internet 信息服务器(IIS)”对话框

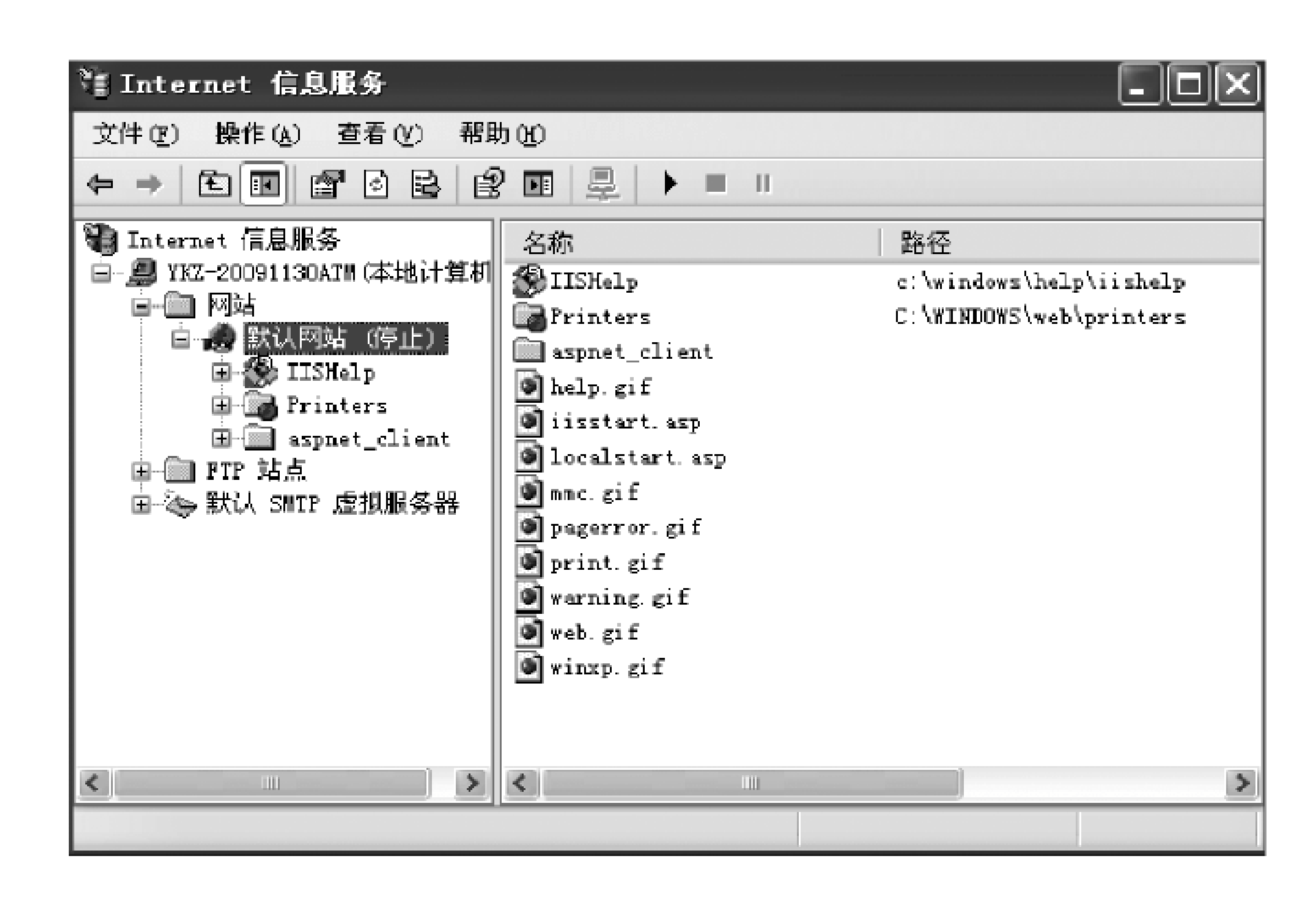


图 2-10 “Internet 信息服务”对话框

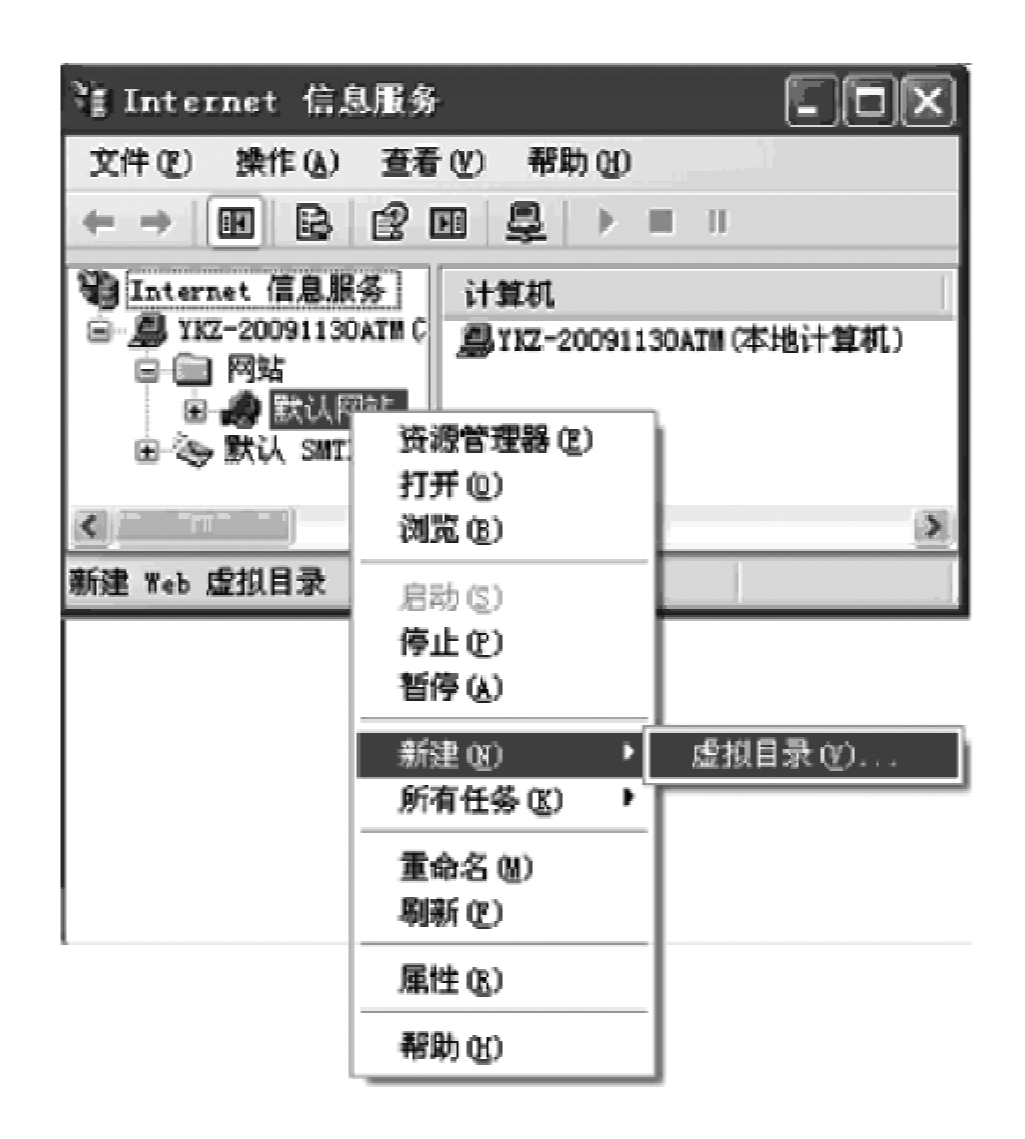


图 2-11 右键快捷菜单

提示：当用户通过HTTP浏览位于Web服务器上的一些Web页面时，Web服务器需要确定与该页面对应的文件位于服务器硬盘上的什么位置。事实上，在由URL给出的信息与包含页面文件的物理位置(在Web服务器的文件系统中)之间有着重要的关系。这个关系是通过虚拟目录来实现。

提示：虚拟目录相当于物理目录在Web服务器机器上的别名，它不仅使用户避免了冗长的URL，也是一种很好的安全措施，因为虚拟目录对所有浏览者隐藏了物理目录结构。

(8) 在硬盘上创建一个物理目录，这里在C盘的根目录下创建一个目录，命名为Sample。

(9) 启动"Internet信息服务"，右击"默认网站"节点，选择"新建"→"虚拟目录"命令，弹出如图2-12所示的"虚拟目录创建向导"对话框。

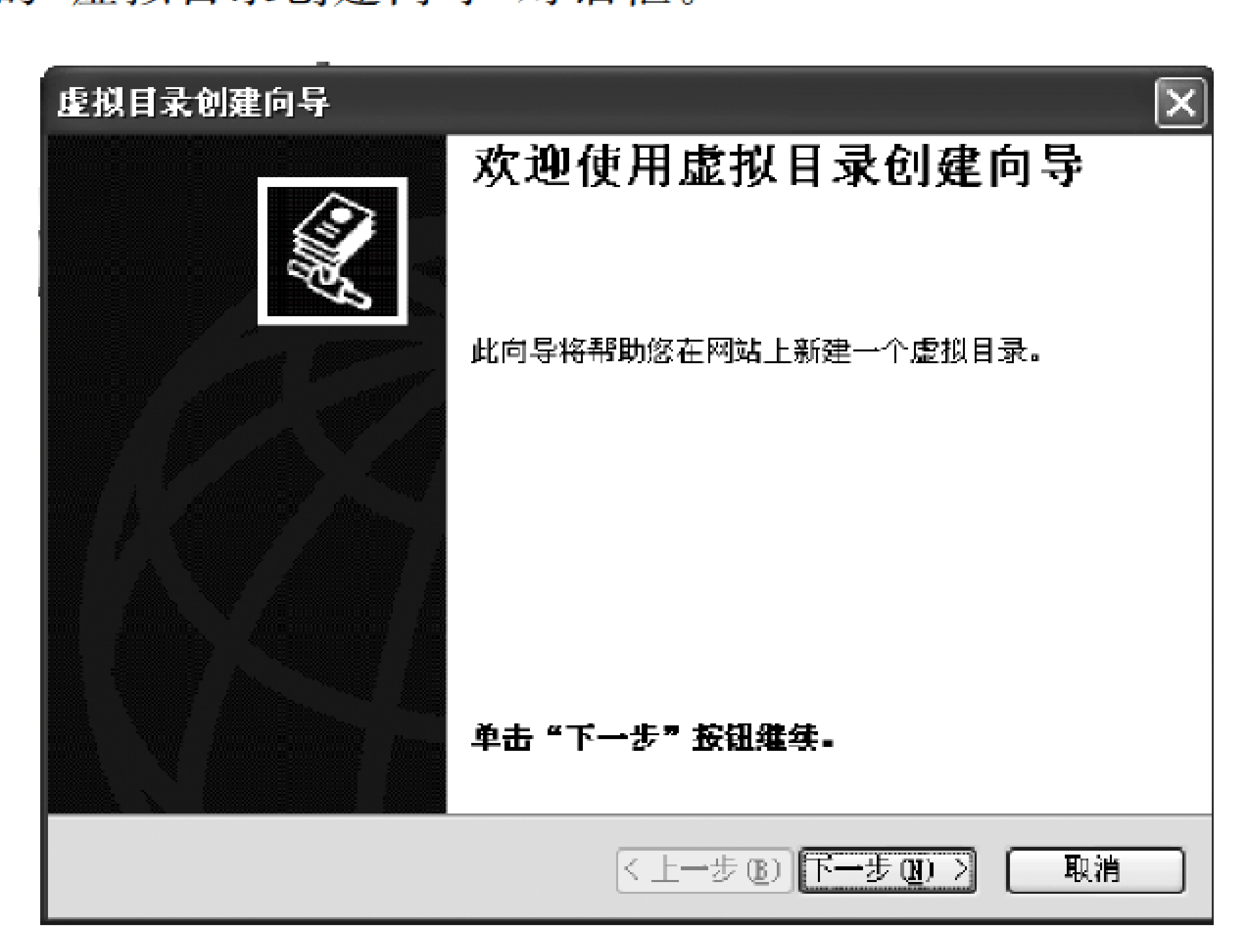


图2-12 "虚拟目录创建向导"对话框

(10) 单击"下一步"按钮，弹出如图2-13所示的"虚拟目录别名"界面。



图2-13 "虚拟目录别名"界面

(11) 在“别名”文本框中输入虚拟目录的名字,这里命名为 Sample,和它的物理目录的名字相同。然后单击“下一步”按钮,弹出如图 2-14 所示的“网站内容目录”界面。

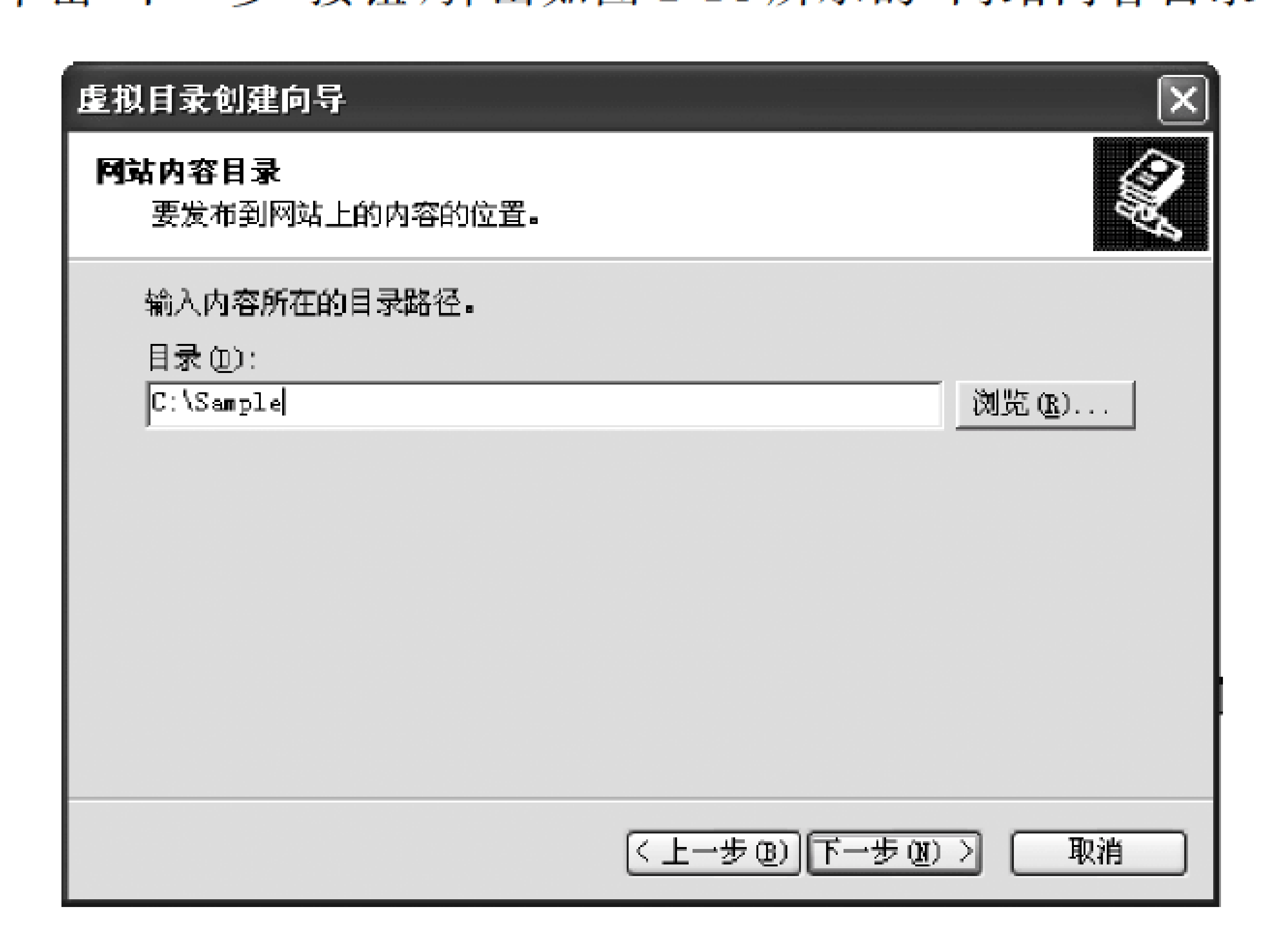


图 2-14 “网站内容目录”界面

(12) 选择刚才创建的物理目录 C:\Sample,单击“下一步”按钮,弹出如图 2-15 所示的“访问权限”界面。

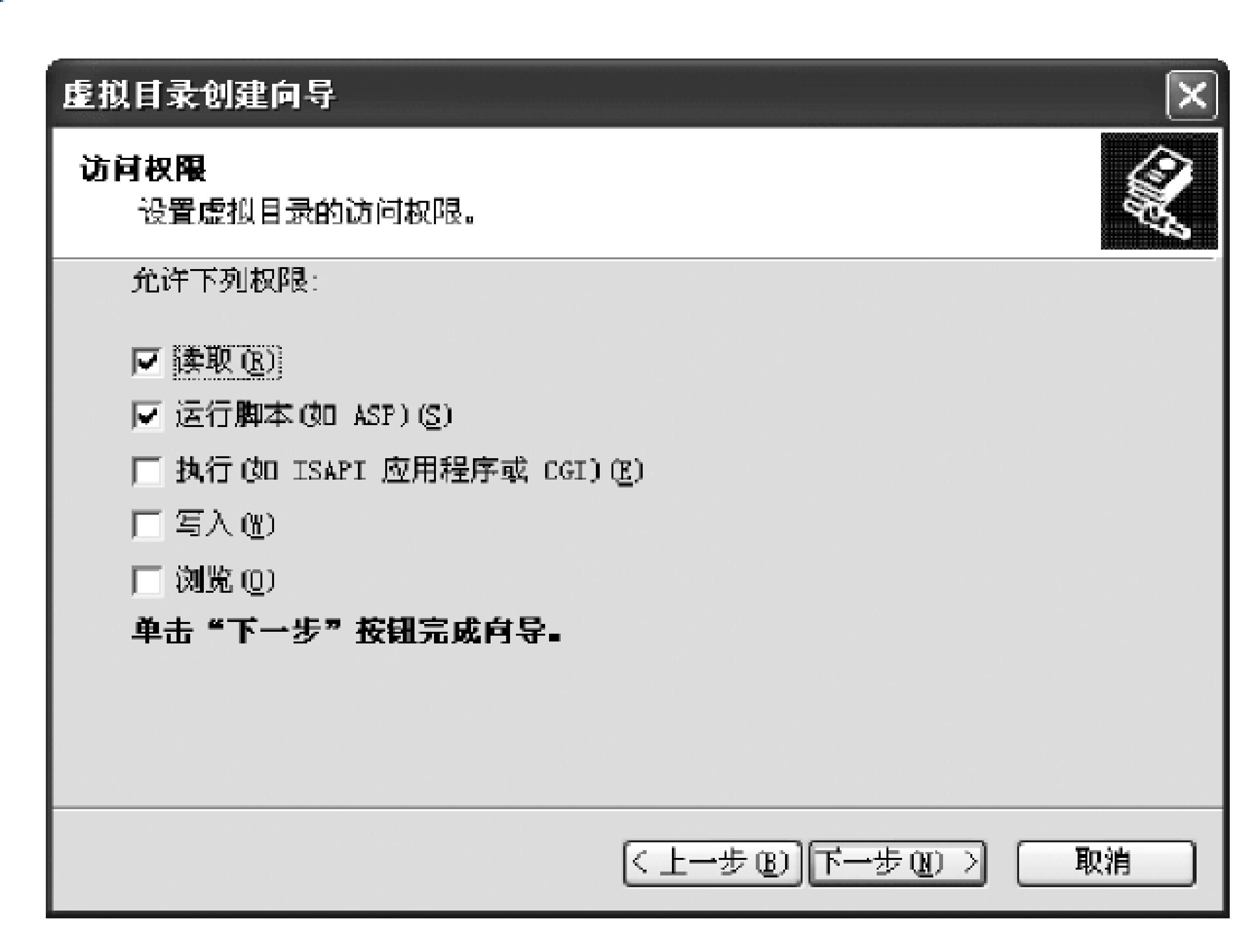


图 2-15 “访问权限”界面

(13) 在“访问权限”界面设置虚拟目录的访问权限,除非读者明白自己需要什么样的权限,否则不要改变创建时默认的权限。单击“下一步”按钮,弹出如图 2-16 所示的“已成功创建虚拟目录”界面。

(14) 单击“完成”按钮,完成虚拟目录的创建。此时,在“Internet 信息服务”窗口的目录中将显示该 Sample 虚拟目录,如图 2-17 所示。

(15) 在 Sample 虚拟目录上右击,从弹出的菜单中选择的“属性”命令,如图 2-18 所示。

(16) 弹出如图 2-19 所示的“Sample 属性”对话框,选择“虚拟目录”选项卡,并设置连接到 Web 站点的内容来源,默认为“此计算机上的目录”,对此目录的默认权限只有“读取”、



图 2-16　“已成功创建虚拟目录”界面

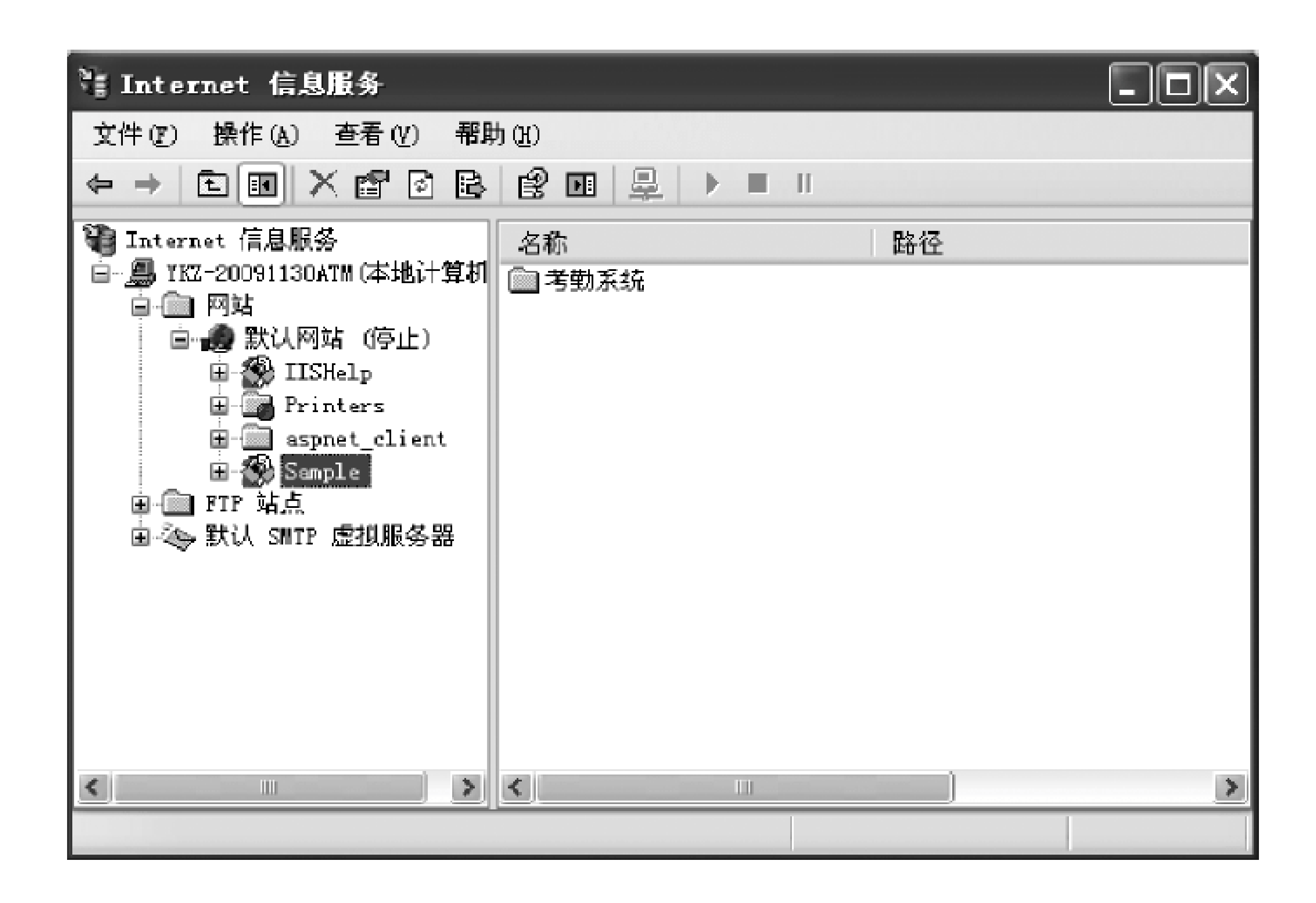


图 2-17　Internet 信息服务

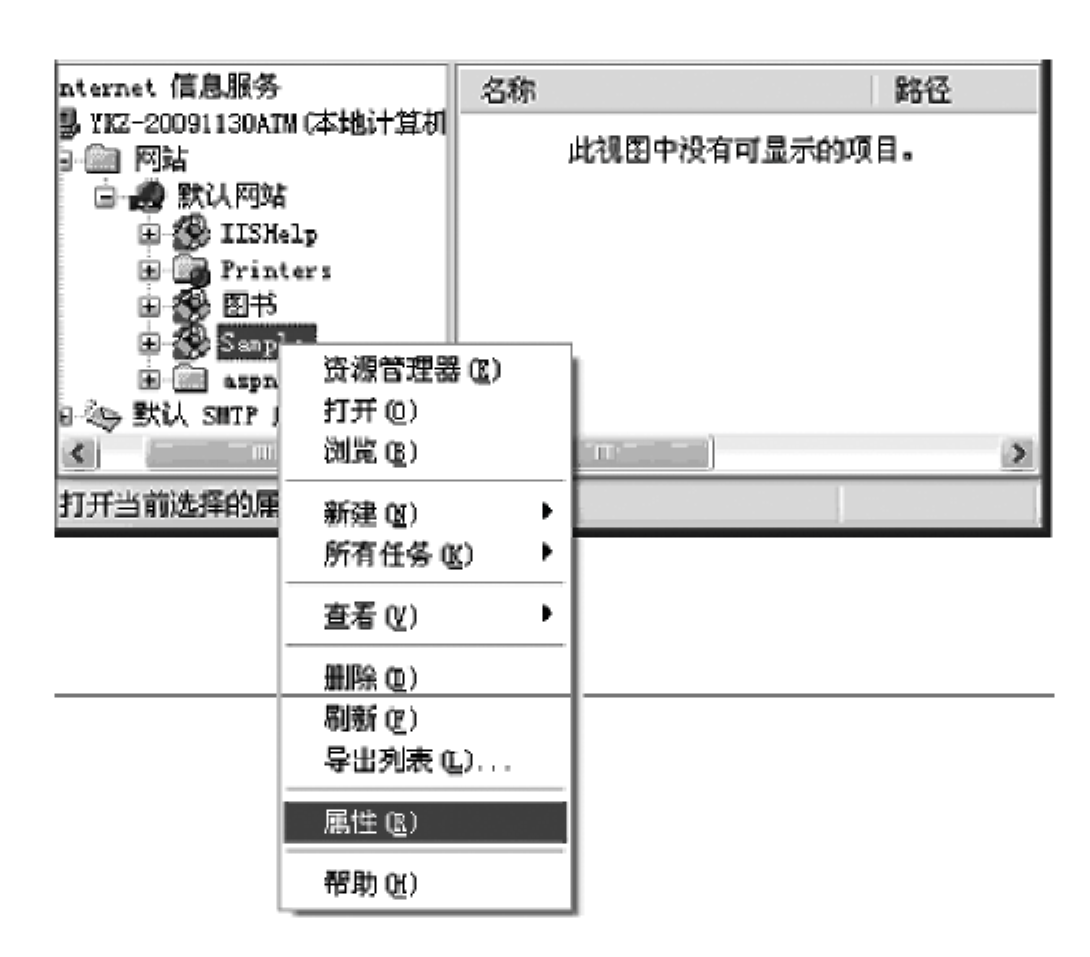


图 2-18　选择“属性”命令

“记录访问”和“索引资源” 3 项，如果没有特殊要求，此选项卡中的内容不需要改动。



图 2-19 “虚拟目录”选项卡

(17) 选择“文档”选项卡，并选中“启动默认文档”复选框，这样当运行 Web 程序后，不需要在地址栏中填写此文件名，系统会读取默认文档中的文件。用户可以添加和删除默认文档，如图 2-20 所示。



图 2-20 “文档”选项卡

(18) 选择“目录安全性”选项卡，设置目录安全性，如图 2-21 所示。共有 3 种方法可以控制目录的安全性，分别为“身份验证和访问控制”、“IP 地址和域名限制”以及“安全通信”，通过这 3 种方法可以有效地控制目录的安全性。



图 2-21　“目录安全性”选项卡

(19) 选择 ASP.NET 选项卡，如图 2-22 所示，设置用户使用的 ASP.NET 版本，这里设置为 2.0.50727，最后单击“确定”按钮，完成所有的 Web 服务器的设置。



图 2-22　ASP.NET 选项卡

任务 2-2-2　安装 Visual Studio 2010

【任务描述】

安装 Visual Studio 2010，搭建 ASP.NET Web 应用程序的集成开发环境。

【任务实施】

（1）单击 Visual Studio 2010 中文旗舰版的 setup.exe 安装程序启动安装。

（2）打开安装程序后，首先进入如图 2-23 所示的安装向导界面。



图 2-23　Visual Studio 2010 安装向导界面

（3）选择"安装 Microsoft Visual Studio 2010"选项，即进入 Visual Studio 2010 的安装，如图 2-24 所示。



图 2-24　加载安装组件的过程

注意：在安装 Visual Studio 2010 之前，Visual Studio 2010 安装程序首先会加载安装组件，这些组件为 Visual Studio 2010 的顺利安装提供了基础保障，安装程序在完成组件的加载之前，用户不能进行安装步骤的选择。

(4) 安装组件加载完毕后，单击"下一步"按钮进入安装程序起始页，选中"我已阅读并接受许可条款"单选按钮，如图 2-25 所示。

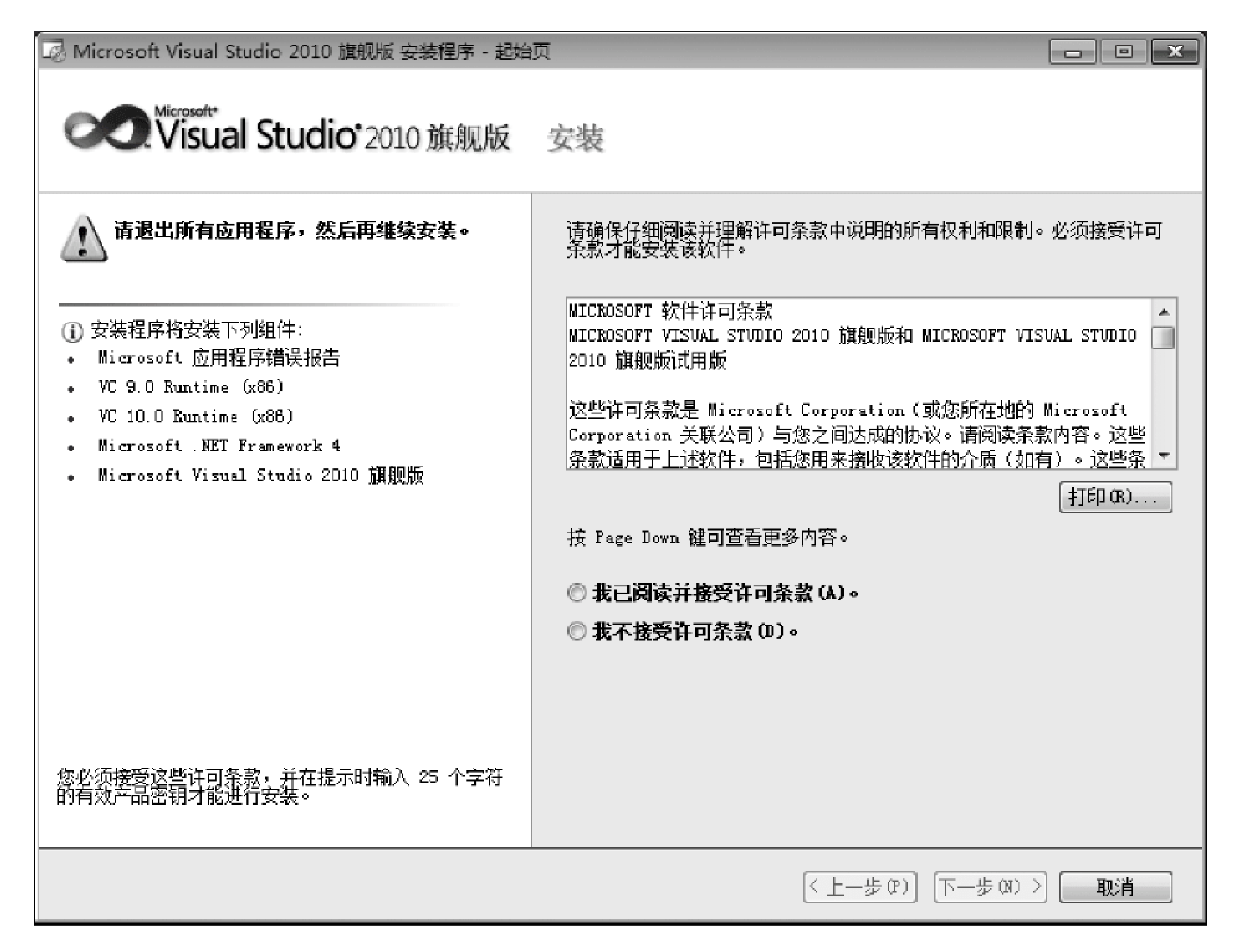


图 2-25　安装程序起始页

(5) 单击"下一步"按钮，进行 Visual Studio 2010 的安装，用户可以设置 Visual Studio 2010 的安装路径，如图 2-26 所示。

注意：当选择安装路径后，就能够进行 Visual Studio 2010 的安装了。在选择路径前，可以选择相应的安装功能，通过选择"完全"或"自定义"单选按钮。选择"完全"将安装 Visual Studio 2010 的所有组件，单击"安装"按钮就开始安装，如图 2-27 所示。而如果只需要安装几个组件，则可以选择"自定义"进行组件的选择安装。

图 2-27 中的安装界面的左侧将显示安装列表的进度，当所有组件安装成功后，进入图 2-28 所示界面，显示已经成功安装 Visual Studio 2010，最后单击"完成"按钮，结束安装过程。至此，Visual Studio 2010 成功地被安装到本地计算机上。

任务 2-2-3　创建简单的 Web 网站

【任务描述】

- 在 Visual Studio 中创建 ASP.NET 网站项目。



图 2-26　Visual Studio 2010 安装路径的选择



图 2-27　Visual Studio 2010 的安装



图 2-28　安装完成界面

- 在客户浏览器中运行默认生成页 Default.aspx，输出用户登录时间。

【任务实施】

1. 创建网站

（1）运行 Visual Studio 2010，在 VS 2010 菜单栏中选择“文件”→“新建网站”命令，打开“新建网站”对话框。

（2）在“新建网站”对话框的左侧，选择“Visual C#”；在对话框中间部分，选择“ASP.NET 网站”；在对话框下方的下拉列表框中，选择“Web 位置”为“文件系统”；单击“浏览”按钮选择站点路径“G:\ASP.NET 网站开发项目化教程\chapter02\ch02_3”，这里的 ch02_3 是应用程序（网站）名称，如图 2-29 所示。

提示：打开“新建网站”对话框的方法不止一种，除了使用如前所述的菜单之外还有其他方法，读者可以尝试，在此不予赘述。

（3）在“新建网站”对话框中单击“确定”按钮，Visual Studio 2010 会自动创建并配置新建的网站 ch02_3，并且会自动创建一个 Web 页面 Default.aspx，同时在主窗体中显示默认文档 Default.aspx 的代码结构，如图 2-30 所示。

提示：在“解决方案资源管理器”窗口中可以看到创建的网站中包括 Default.aspx、Default.aspx.cs、web.config 文件和文件夹 App_Data、Scripts。其中 Default.aspx 文件是所创建的 Web 页面，扩展名为.aspx，Default.aspx.cs 文件是 Web 页面文件 Default.aspx 对应的程序代码文件，web.config 文件是网站配置文件，文件夹 App_Data 是存放网站数据

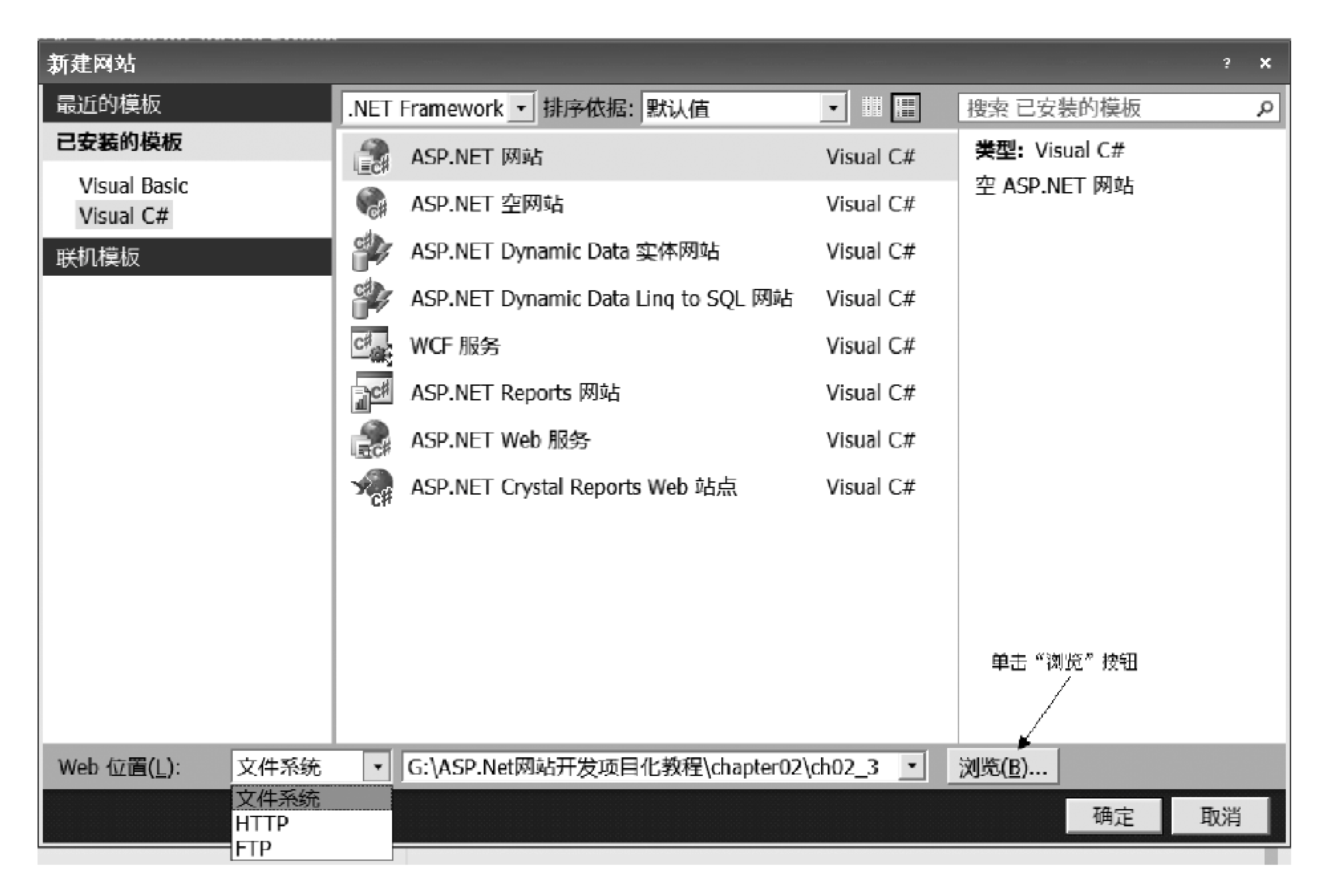


图 2-29 “新建网站”对话框

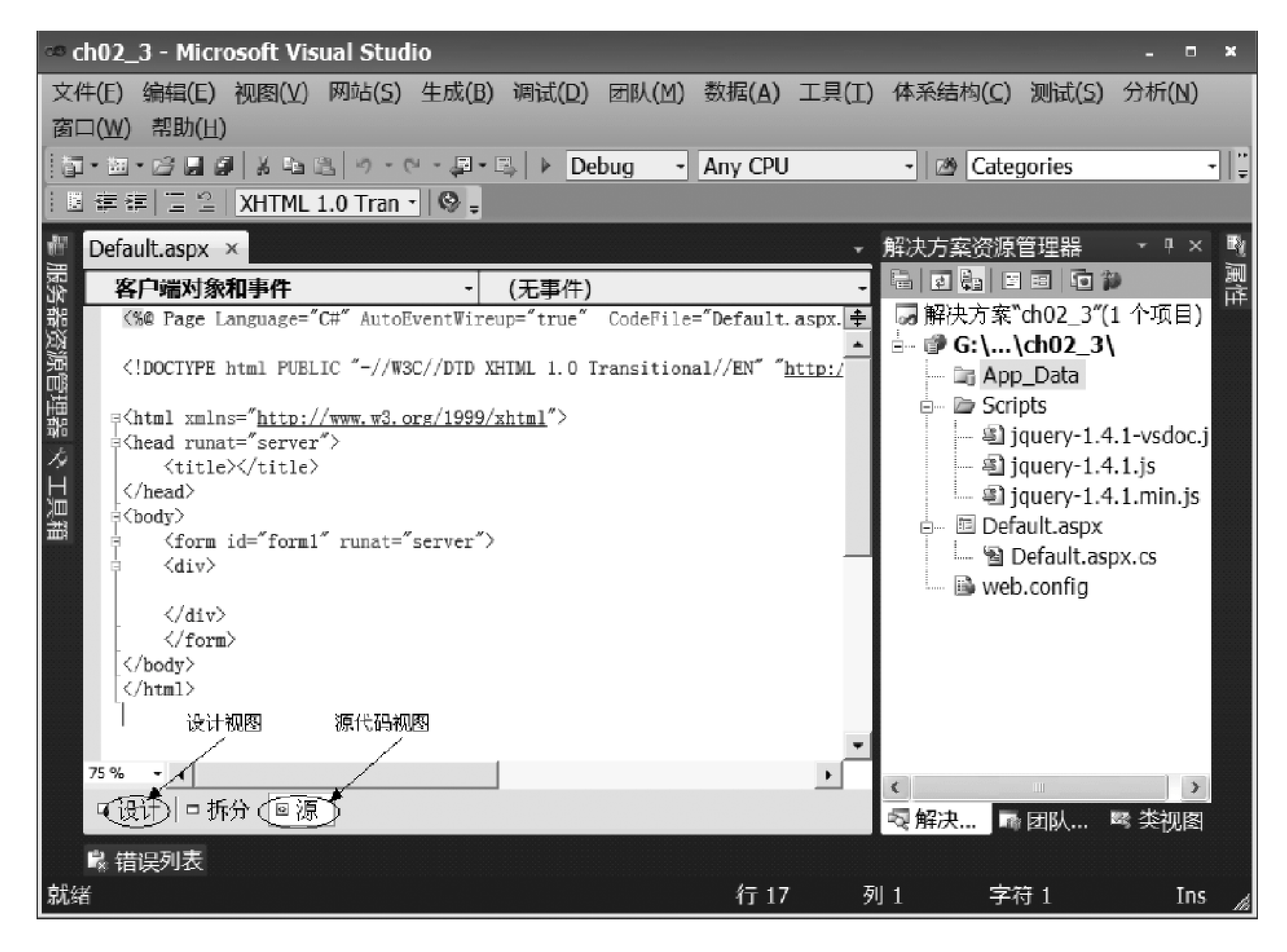


图 2-30 Visual Studio Web 窗体界面

库的文件夹，Scripts 用于存放网站 JS 脚本文件，实际开发过程中用户还可以根据需要自己创建文件夹用于存放应用程序（网站）所需要的其他资源，如创建 images 文件夹用于存放网站图片。

2. 设计 Web 页面

（1）在页面代码视图中＜title＞＜/title＞之间输入网页标题“新知书店网”。

（2）单击编辑窗体底部的“设计视图”按钮，切换到网页的设计视图，在 div 区域输入文字“欢迎您光临新知书店网”，如图 2-31 所示，然后单击工具栏中的“保存”按钮或“全部保存”按钮保存新建的页面。

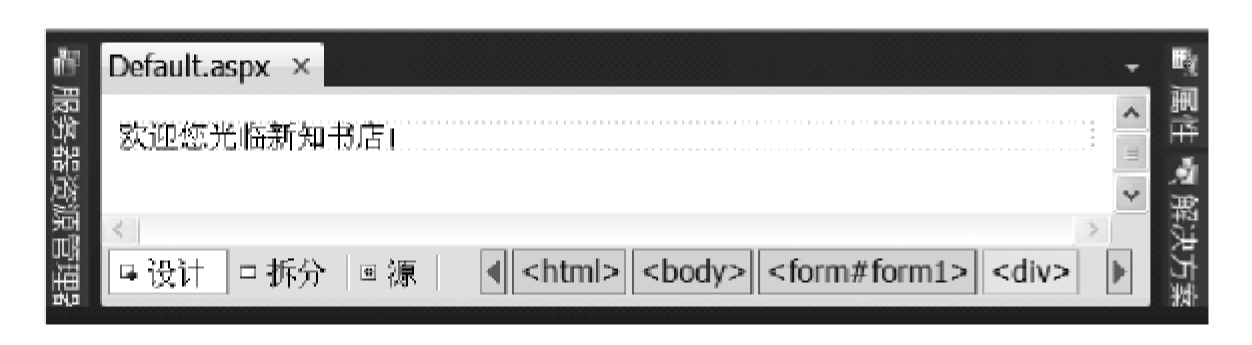


图 2-31　Web 页中输入文字

（3）单击编辑窗口底部的“源”按钮，切换到页面的源代码视图，如图 2-32 所示。

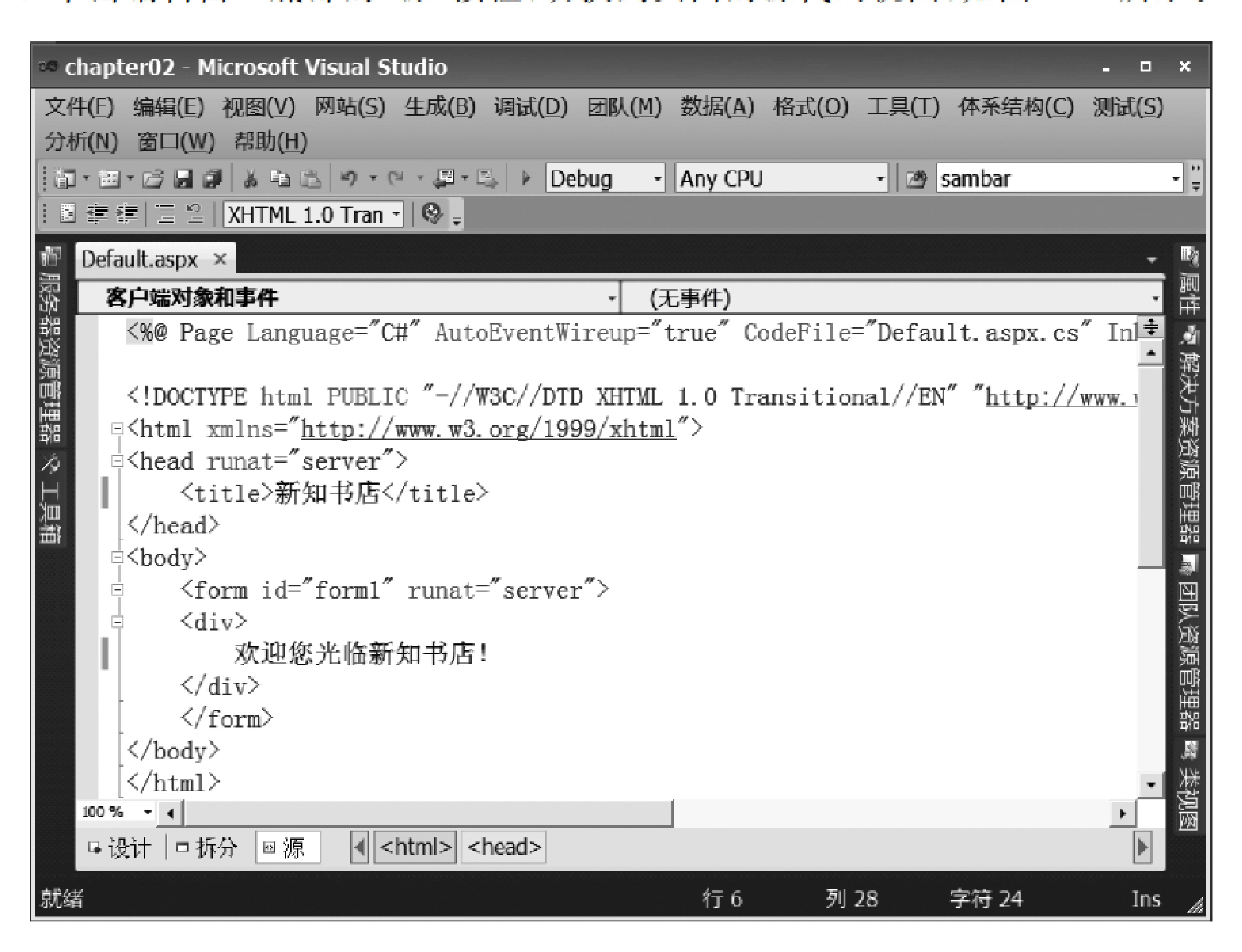


图 2-32　Web 页面的源代码视图

3. 编写程序代码

在“解决方案资源管理器”窗口中双击 Default.aspx.cs，切换到程序逻辑代码编写页面，在逻辑代码编写页面 Default.aspx.cs 的代码编辑区域中为 Page 对象的 Load 事件编写功能代码，输出用户登录时间，如图 2-33 所示。然后单击工具栏中的“保存”按钮或“全

部保存"按钮 保存 Default.aspx.cs 页中输入的程序逻辑代码。

```
Default.aspx.cs × Default.aspx
_Default                                  Page_Load(object sender, EventArgs e)
 1 using System;
 2 using System.Collections.Generic;
 3 using System.Linq;
 4 using System.Web;
 5 using System.Web.UI;
 6 using System.Web.UI.WebControls;
 7
 8 public partial class _Default : System.Web.UI.Page
 9 {
10     protected void Page_Load(object sender, EventArgs e)
11     {
12         Response.Write("您的登录时间为: " + DateTime.Now + "");
13     }
14 }
100 %
```

图 2-33　在 Web 页中输入文字

图 2-33 中 Page 对象的 Load 事件的逻辑代码(即第 12 行)如下:

```
Response.Write("您的登录时间为: " + DateTime.Now + "");
```

其余代码均为系统自动生成。

4. 预览页面

直接按 F5 键或者在主窗口中选择"调试"→"启动调试"命令浏览网页,如果出现了如图 2-34 所示的"未启用调试"对话框,直接单击"确定"按钮,自动添加启用调试的 web.config 配置文件,激活调试功能。页面的浏览效果如图 2-35 所示。



图 2-34　"未启用调试"对话框

任务 2-2-4　分析 ASP.NET 文档

【任务描述】

- 认识 ASP.NET 文档的内容和结构。



图 2-35　Web 页面 Default.aspx 的预览效果

• 分析 ASP.NET 页面的功能代码。

【任务实施】

1. 打开任务 2-2-3 中所创建的网站项目 ch02_3

启动 Visual Studio 2010，在主窗口中选择“文件”→“打开网站”命令，在“打开网站”对话框中选择任务 2-2-3 所创建的网站项目为 ch02_3 的文件夹，该网站夹的路径为“G:\ASP.NET 网站开发项目化教程\chapter02\ch02_3”，然后单击“打开网站”对话框中的“打开”按钮，即可打开网站 ch02_3，如图 2-36 所示。

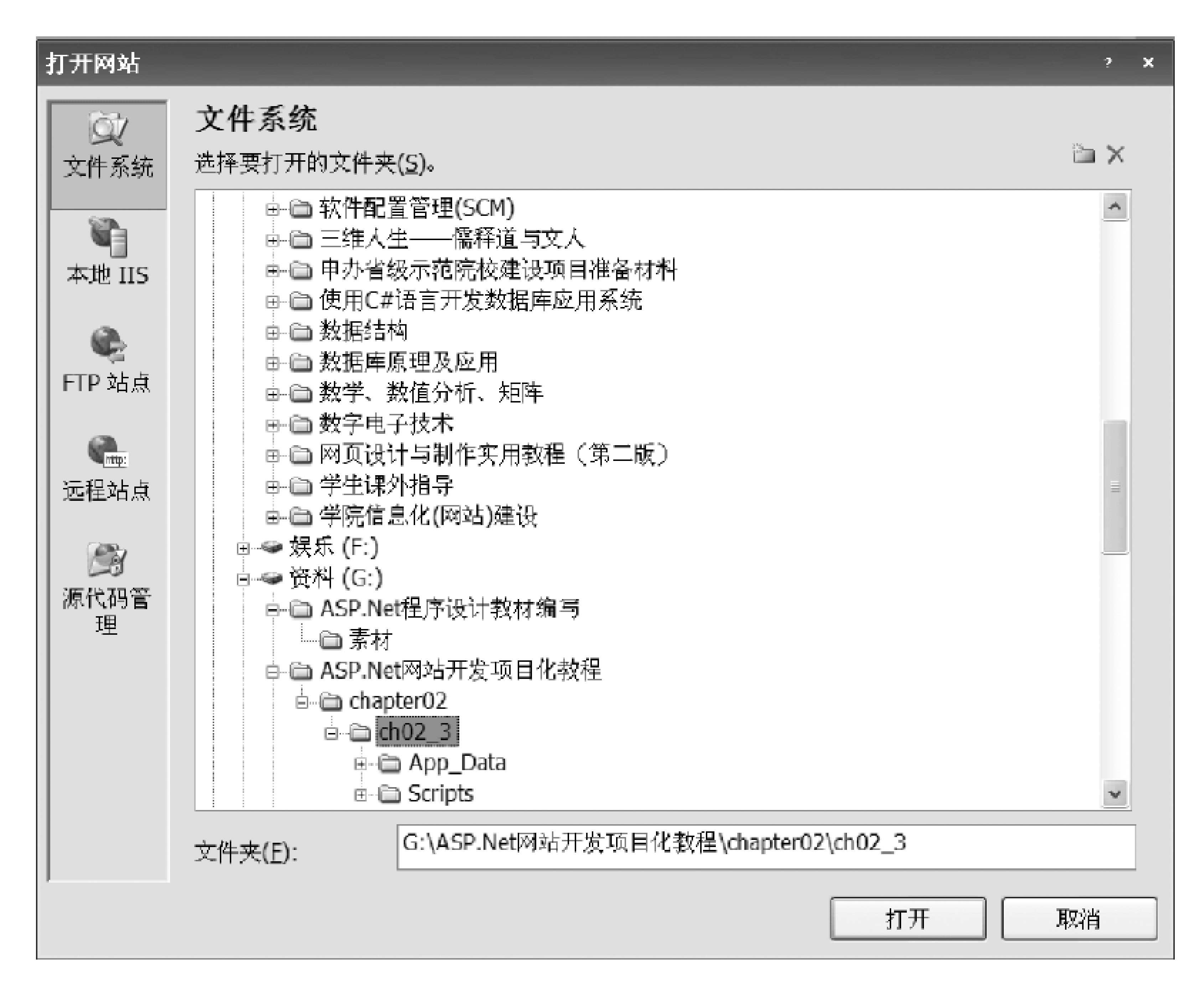


图 2-36　“打开网站”对话框

2. 分析 Web 页面的源代码

任务 2-2-3 中所创建网站 ch02_3 的 Web 页面 Default.aspx 的源代码如表 2-1 所示。

表 2-1　网站 ch02_3 的 Web 页面 Default.aspx 的源代码

行号	代　码　页
01	<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
02	Inherits="_Default" %>
03	
04	<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
05	"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
06	
07	<html xmlns="http://www.w3.org/1999/xhtml">
08	<head runat="server">
09	<title>新知书店网</title>
10	</head>
11	<body>
12	<form id="form1" runat="server">
13	<div>
14	欢迎您光临新知书店网!
15	</div>
16	</form>
17	</body>
18	</html>

表 2-1 的代码解释：

(1) 01-02 行代码为页面指令，即“<%@ Page %>”指令，主要为 ASP.NET 页面文件制定解析和编译时使用的属性和值，每一个.aspx 页面文件只能包含一条@ Page 指令。其中，@ Page 指令的 Language 属性为网页文档指定程序语言类型；AutoEventWireup 属性的默认值为 true，表示将自动调用页面事件(相关知识将在后续章节中详述)；CodeFile 指定了与页面相关的后置代码文件，本例后置代码文件为 Default.aspx.cs；Inherits 属性定义了供页面继续的代码后置的类。

(2) 04-05 行声明文档的类型，说明网页文档使用的 XHTML 是哪一个版本，这里使用的是 XHTML 1.0。

(3) 07-18 行的 HTML 源代码被分为两部分：<head>…</head>之间的网页头部区域和<body>…</body>之间的网页主体部分。

(4) <form>…</from>表示网页中包含一个表单对象。

(5) <div>…</div>表示布局区块，div 是一种 XHTML 的布局标签。

(6) 第 08 行和 12 行所出现的“runat="server"”指明该代码在服务器端执行。

3. 分析 Web 页面的程序逻辑(功能)代码

任务 2-2-3 中所创建的 Web 页面 Default.aspx 引用的代码隐藏文件的程序逻辑(功能)代码如表 2-2 所示。

表 2-2　Web 页面 Default. aspx 的代码文件 Default. aspx. cs 中的代码

行号	代　码　页
01	using System;
02	using System.Collections.Generic;
03	using System.Linq;
04	using System.Web;
05	using System.Web.UI;
06	using System.Web.UI.WebControls;
07	
08	public partial class _Default : System.Web.UI.Page
09	{
10	protected void Page_Load(object sender, EventArgs e)
11	{
12	Response.Write("您的登录时间为: " + DateTime.Now + "");
13	}
14	}

表 2-2 的代码解释：

(1) 01～06 行表示引入的多个命名空间。

(2) 08～14 行表示创建的类，10～13 行表示页面对象 Page 的 Load 事件过程的程序代码。

(3) 12 行表示在页面输出您登录的时间，DateTime 是一个结构类型，Now 是 DateTime 的属性，用于获取当前系统日期和时间。

2.3　知识拓展

2.3.1　ASP.NET 页面的处理机制

ASP.NET 页面由. aspx 文件和. cs 文件构成，事实上. cs 文件和. aspx 文件中标有 runat＝“server”属性(该属性会在后续内容中介绍)的元素被编译成一个类，两者是局部类(由关键字 partial 声明的类)的关系，在运行过程中，可以将 Web 页面的. cs 文件和. aspx 文件看成一个整体，Web 页面的处理过程如下：

(1) 用户通过客户端浏览器请求页面，页面第一次运行。如果程序员通过编程让它执行初步处理，如对页面进行初始化操作等，可以在 Page_load 事件中进行处理。

(2) Web 服务器在其硬盘中定位所请求的页面。

(3) 如果 Web 页面的扩展名为. aspx，就把这个文件交给 aspnet-isapi. dll 进行处理。如果以前没有执行过这个程序，那么就由 CLR 编译并执行，得到纯 HTML 结果；如果已经执行过这个程序，那么就直接执行编译好的程序并得到纯 HTML 结果。

(4) 把 HTML 流返回给浏览器，浏览器解释执行 HTML 代码，显示 Web 页面的内容。

上述过程可以用图 2-37 加以说明。

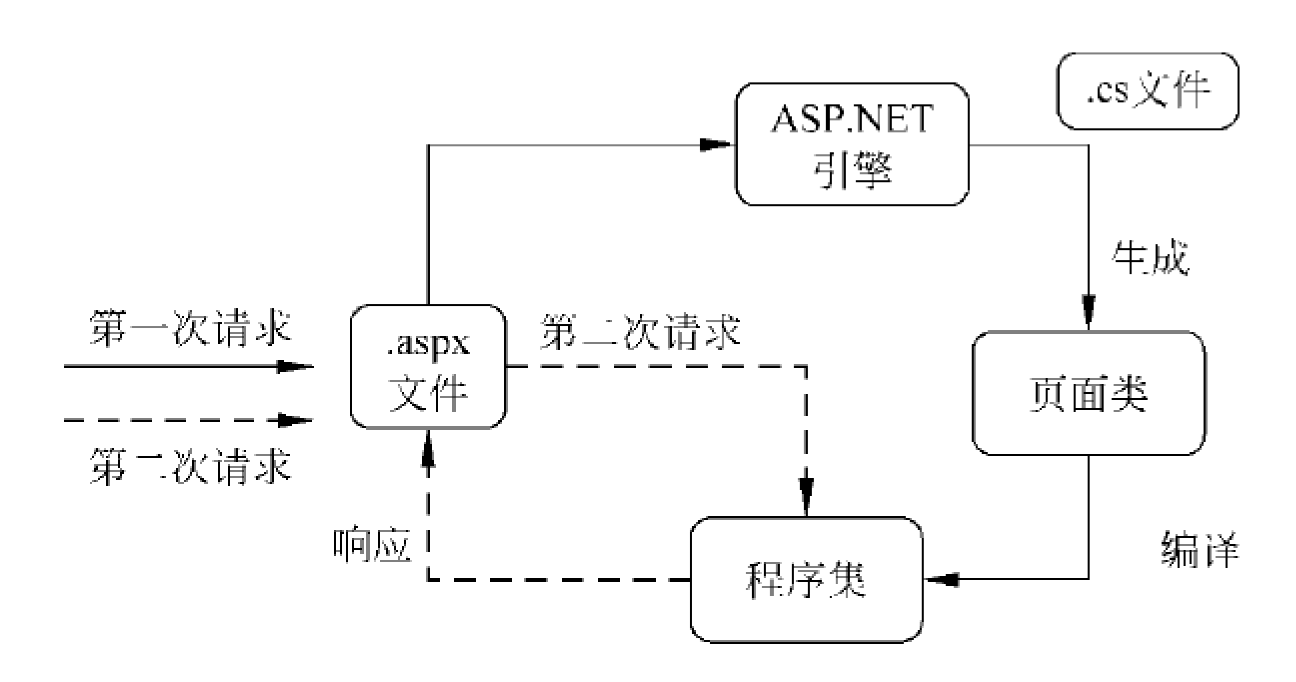


图 2-37　ASP.NET 页面运行机制

2.3.2　ASP.NET 的网页代码模型

ASP.NET 网页由以下两部分组成：

- 可视元素，包括标记、服务器控件和静态文本。
- 页的编程逻辑，包括事件处理程序和其他代码。

ASP.NET 提供了两个用于管理可视元素和代码的模型，即单文件页和代码隐藏页模型。这两个模型功能相同，两种模型中可以使用相同的控件和代码。

1. 单文件页模型

在单文件页模型中，页的标记及其编程代码位于同一个后缀为.aspx 的文件中。可以通过下面的操作创建一个单文件页模型，打开任务 2-2-3 所创建的网站项目 ch02_3，在“解决方案资源管理器”窗口中右击网站 ch02_3，从弹出的快捷菜单中选择“添加新建项”命令，在“名称”文本框中输入 sPage.aspx，如图 2-38 所示，取消选中“将代码放在单独的文件中”



图 2-38　创建单文件页模型

复选框，单击“添加”按钮，即可创建单文件页模型的 ASP.NET 文件，创建后会自动创建相应的 Html 代码以便页面的初始化，示例代码如表 2-3 所示。

表 2-3　单文件页模型的 sPage.aspx 页面代码

行号　代　码　页

```
<%@ Page Language="C#" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<script runat="server">

</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
</html>
```

表 2-3 中的代码演示了一个单文件页。业务逻辑代码位于<script>…</script>标记的模块中，以便与其他显示代码隔离开。服务器端运行的代码一律在<script>标记中注明 runat="server"属性，此属性将其标记为 ASP.NET 应执行的代码。一个<script>模块可以包括多个程序段，每个网页也可以包括多个<script>模块。

<script runat="server">中的 runat 是<script>标记的一个属性，属性值为"server"，表示<script>块中包含的代码在服务器端而不是客户端运行，此属性对于服务器端代码是必需的。

在对单文件页进行编译时，编译器将生成并编译一个从 Page 基类派生或从使用@Page 指令的 Inherits 属性定义的自定义基类派生的新类。在生成页之后，生成的类将编译成程序集，并将该程序集加载到应用程序域，然后对该页类进行实例化并执行该页类，以将输出呈现到浏览器。单文件页模型如图 2-39 所示。

2. 代码隐藏页模型

在创建网页时，如果选中“将代码放在单独的文件中”复选框，即可创建代码隐藏页模型的 ASP.NET 文件。代码隐藏页模型与单文件页模型不同的是，代码隐藏页模型将事物处理代码都存放在.cs 文件中，当 ASP.NET 网页运行的时候，ASP.NET 类生成时会先处理.cs 文件中的代码，再处理.aspx 页面中的代码。这种过程被称为代码分离。

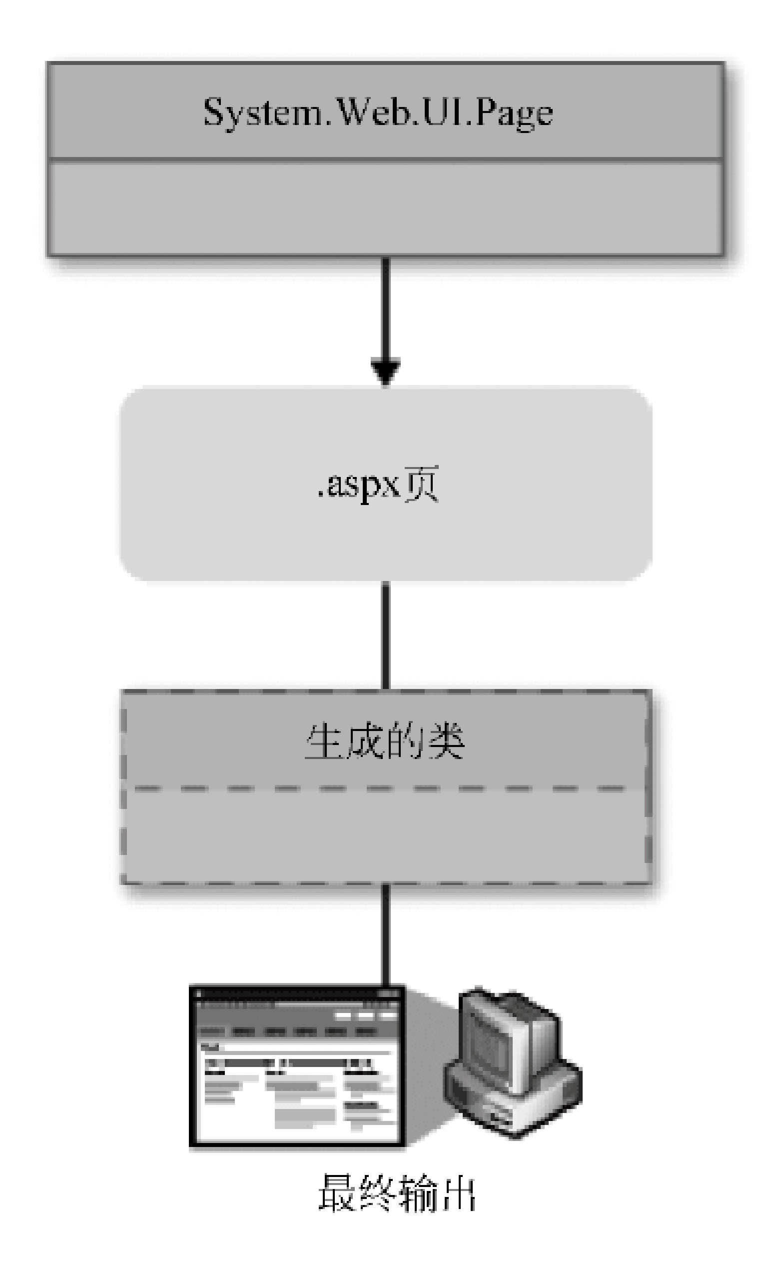


图 2-39　单文件页模型

代码分离有一种好处，就是在.aspx 页面中，开发人员可以将页面直接作为样式来设计，即美工人员也可以设计.aspx 页面，而.cs 文件由程序员来完成事务处理。同时，将 ASP.NET 中的页面样式代码和逻辑处理代码分离能够让维护变得简单，同时代码看上去也非常的优雅。在.aspx 页面中，代码隐藏页模型的.aspx 页面代码基本上和单文件页模型的代码相同，不同的是在 script 标记中的单文件页模型的代码默认被放在了同名的.cs 文件中，任务 2-2-3 创建网站 ch02_3 时自动生成的 Web 页面 Default.aspx 就是一个典型的代码隐藏页模型。

与单文件页模型相比，代码隐藏页模型的.aspx 页有两处差别。在代码隐藏模型中，不存在具有 runat="server"特性的 script 块(如果要在页中编写客户端脚本，则该页可以包含不具有 runat="server"特性的 script 块)。第二个差别是，代码隐藏模型中的@ Page 指令包含引用外部文件(如 Default.aspx.cs)和类的特性，如多态、继承等。这些特性将.aspx 页链接至其代码。

代码隐藏页模型的优点包括以下几点：

- 适用于包含大量代码或多个开发人员共同创建网站的 Web 应用程序。
- 代码隐藏页可以清楚地分隔标记(用户界面)和代码。这一点很实用，可以在程序员编写代码的同时让设计人员处理标记。
- 代码并不会向仅使用页标记的页设计人员或其他人员公开。
- 代码可在多个页中重用。

但是 ASP.NET 代码隐藏页模型的运行过程比单文件页模型要复杂，运行示例图如图 2-40 所示。

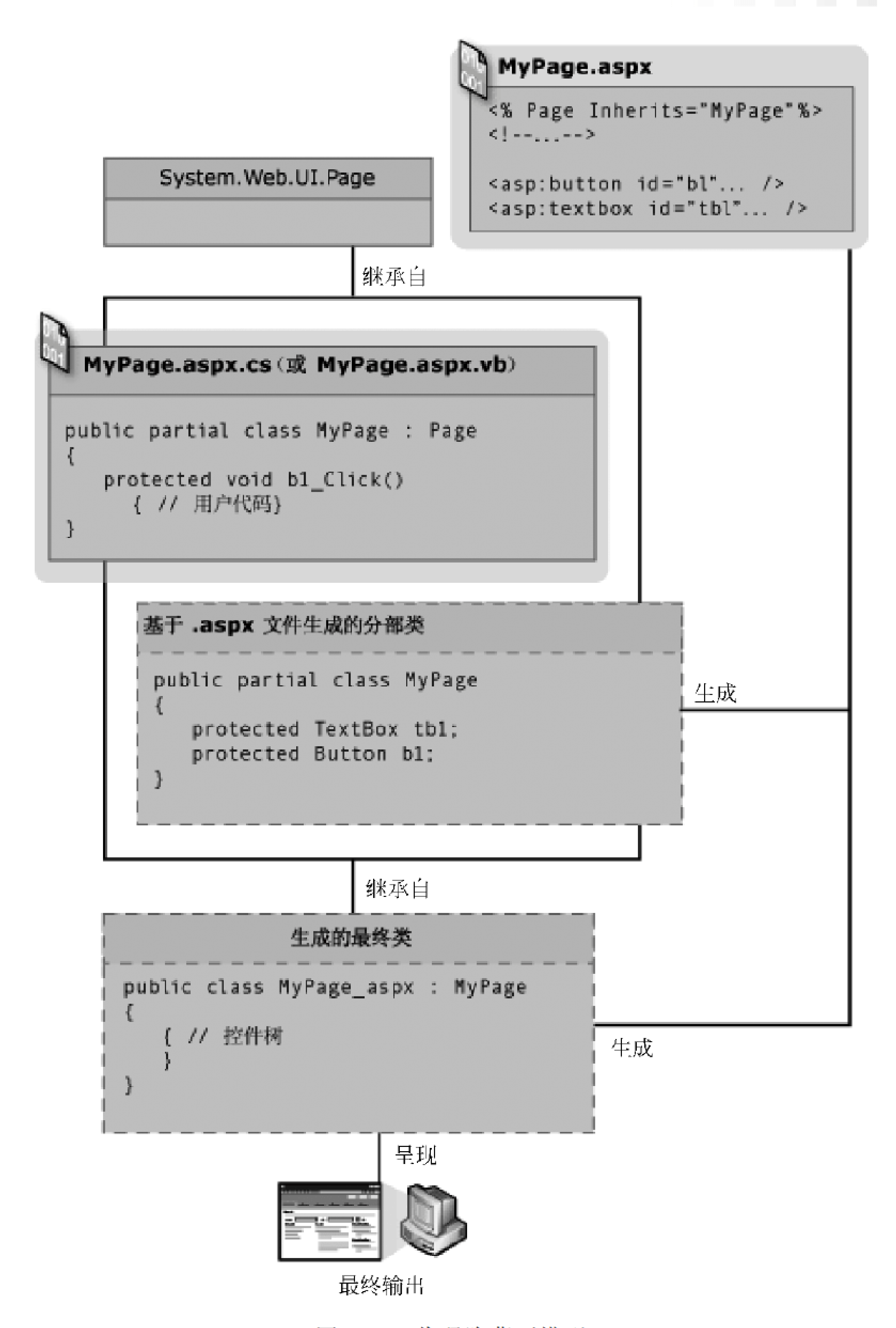


图 2-40　代码隐藏页模型

2.4　项目实训

1. 搭建网站开发环境

【需求说明】

- 安装并配置 IIS 服务器。
- 安装并熟悉 Visual Studio 2010 开发环境。

2. 创建“博客系统”的 Web 网站

【需求说明】

- 在 Visual Studio 中创建“博客系统”的网站项目 MyBlog。
- 在创建的网站项目 MyBlog 中添加一个 Web 页面，输出姓名、邮箱账号、QQ 号等个人信息和时间。

2.5 单元小结

本单元首先介绍了.NET Framework 及 Web 程序设计的一些基础知识，如 HTTP 协议的工作方式、服务器和浏览器的概念、B/S 开发模式，然后对静态网页和动态网页的工作原理进行了分析和比较。接着从 ASP.NET 的历史、ASP.NET 的优点等方面对 ASP.NET 技术进行了简单的介绍，并介绍了 ASP.NET 开发环境的获取和安装，为用户进一步学习奠定了基础。接下来讲解了 ASP.NET 页面是如何组织和运行的，包括页面的往返与运行(处理)机制，在了解这些基本运行机制后，就能够在.NET 框架下进行 ASP.NET 开发了。最后，介绍了在编写 ASP.NET 网页时所采用的代码模型，代码模型有单文件页模型和代码隐藏页模型，在单文件页模型中，页的标记及其程序代码位于同一个后缀名为.aspx 的文件中，而在代码隐藏页模型中，页的标记和服务器端元素仍位于.aspx 文件中，程序代码则单独位于后缀名为.cs 的代码隐藏文件中。本单元的知识体系如图 2-41 所示。

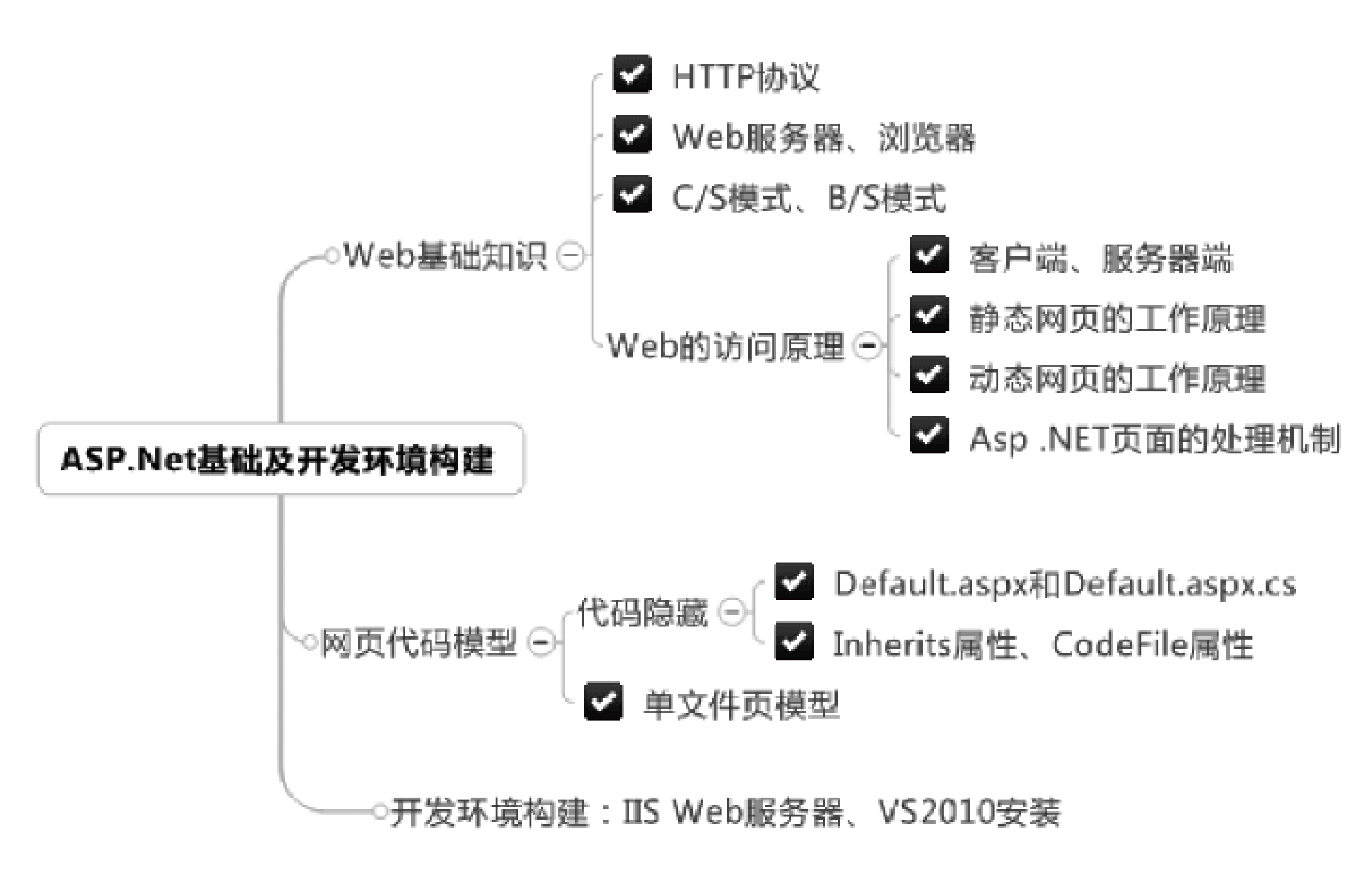


图 2-41 ASP.NET 基础及开发环境搭建知识体系

2.6 单元练习题

一、选择题

1. 下面哪一个不是动态网页技术(　　)。

A. ASP.NET　　B. ASP　　C. JSP　　D. HTML

2. 可以不用发布就能在本地计算机上浏览的页面编写语言是(　　)。

A. ASP　　B. HTML　　C. PHP　　D. JSP

3. 默认的 ASP.NET 页面文件扩展名是(　　)。

A. ASP　　B. ASPNET　　C. Net　　D. ASPX

4. 关于 Web 服务器,下列描述不正确的是(　　)。

A. 互联网上的一台特殊机,给互联网的用户提供 WWW 服务

B. Web 服务器上必须安装 Web 服务器软件

C. IIS 是一种 Web 服务器软件

D. 当用户浏览 Web 服务器上的网页的时候,是使用 C/S 的工作方式。

5. 在 IIS 的默认网站下创建了一个 chapter1 虚拟目录,如果想访问该目录下的 1_1. htm 页面,下面(　　)是正确的。

A. http://localhost/chapter1

B. http://localhost/asp. net/chapter1

C. http://localhost/chapter1/1_1. htm

D. /chapter1/1_1. htm

6. 如果外地朋友通过 Internet 访问你的计算机上的 ASP.NET 文件,应该选择(　　)。

A. http://localhost/chapter1/1-1. aspx

B. /chapter1/1-1. aspx

C. http://你的计算机名字/chapter1/1-1. aspx

D. http://你的计算机 IP 地址/chapter1/1-1. aspx

7. .NET 框架的核心是(　　)。

A. .NET Framework　　B. IL　　C. FLC　　D. CLR

8. ASP.NET 程序代码编译的时候,.NET 框架先将源代码编译为(　　)。

A. 汇编语言　　B. IL　　C. CS 代码　　D. 机器语言

二、填空题

1. 计算机中安装________以后,系统就可以运行任何.NET 语言编写的软件。

2. .NET Framework 由两部分组成:________和________。

3. CLR 是指________,其功能是负责________。

4. .NET Framework 公共语言运行库最重要的功能是为 ASP.NET 提供________。

5. IIS 是指________。

6. 目前最专业的.NET 开发工具是________。

三、问答题

1. ASP.NET 有哪些优点?

2. 简述什么是.NET 框架。

3. 简述安装和配置 IIS 服务器的步骤。

4. 简述静态网页和动态网页的工作原理。

5. 简述 ASP.NET 页面的处理机制。

单元3 使用控件高效创建网站页面

教学目标：

- 会使用 HTML 服务器控件创建 Web 页面。
- 会使用 Web 标准服务器控件创建 Web 页面。
- 会使用数据验证控件验证 Web 页面中输入的数据。
- 会创建自定义 Web 用户控件。
- 掌握常用 Web 标准服务器控件的功能、属性和事件。
- 掌握常用数据验证控件的功能和属性。
- 了解常用的第三方控件 CKeditor。

3.1 知识准备

3.1.1 服务器控件概述

控件是对数据和方法的封装，也可以理解为是一个可重用的组件或对象，比如人们在网页上经常看到输入信息用的文本框、单选按钮、复选框、下拉列表等元素，都属于控件。控件可以有自己的属性、方法和可以响应的事件。而 ASP.NET 控件是一种服务器端运行的组件，服务器可以根据客户端浏览器的类型将其生成适合在该浏览器运行的 HTML 标记，进而在客户端呈现。

部署在 Web 服务器上的网站，用户可以通过浏览器来访问。当用户请求一个静态的 HTML 页面时，服务器找到对应的文件直接将其发送给用户端的浏览器；而在请求 ASP.NET 页面时(扩展名为.aspx 的页面)，服务器将在文件系统中找到并读取对应的页面，然后将页面中的服务器控件转换成浏览器可以解释的 HTML 标记和一些脚本代码，并将转换后的结果页面发送给用户。

在 ASP.NET 页面上，服务器控件表现为一个标记，如<asp:textbox.../>。这些标记不是标准的 HTML 元素，因此，如果它们出现在网页上，浏览器将无法理解。然而，当从 Web 服务器上请求一个 ASP.NET 页面时，这些标记都将被转换为 HTML 元素，因此浏览器只会接收到它能理解的 HTML 内容。

可以将任意的服务器控件放置在创建的.aspx 页面上，然而请求服务器上该页面的浏览器将只能接收 HTML 和 JavaScript 脚本代码。浏览器无法理解 ASP.NET，因此，Web

服务器须读取 ASP.NET 代码并进行处理，将所有 ASP.NET 特有的内容转换为 HTML 并发送给浏览器进行显示。

1. 控件的种类

启动 Visual Studio 2010 后，选择“视图”→“工具箱”命令，可以看到“工具箱”中有以下控件，Web 控件的关系如图 3-1 所示。

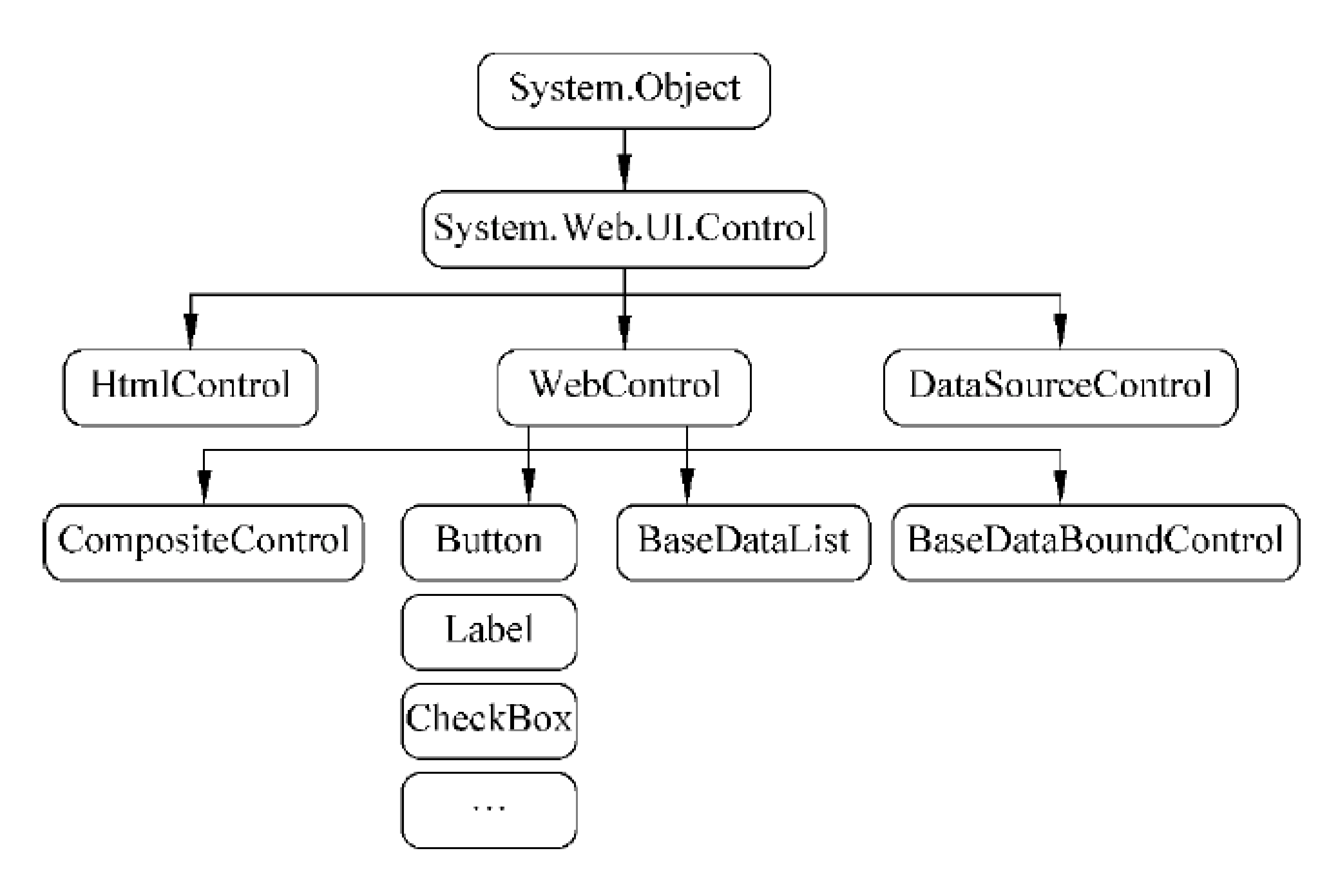


图 3-1 Web 控件关系图

- HTML 服务器控件：提供了对标准 HTML 元素的类封装。在 HTML 控件中添加一个在服务器端运行的属性，即可以由通用的客户端 HTML 控件转变为服务器端 HTML 控件，使开发人员可以对其进行编程。
- Web 服务器控件：比 HTML 服务器控件具有更多功能。Web 服务器控件不仅包括窗体控件（例如按钮和文本框），而且还包括特殊用途的控件（例如日历、菜单和树视图控件）。Web 服务器控件与 HTML 服务器控件相比更为抽象，因为其对象模型不一定反映 HTML 语法。
- 验证控件：这些控件可以使开发人员更容易对一些控件中的数据进行验证。如验证控件可用于对必填字段进行检查，对照字符的特定值或模式进行测试，验证某个值是否在限定范围之内等。
- 导航控件：这些控件被设计用于显示站点地图，允许用户从一个网页导航到另一个网页，如 Menu 控件、SiteMapPath 控件等。
- 数据控件：用于显示大量数据的控件，如 GridView、ListView 控件等，这些控件支持很多高级的定制功能，比如使用模板、允许添加、删除、编辑等。数据控件还包括数据源控件，如 SqlDataSource、LinqDataSource 控件等，使开发人员能够使用声明的方式绑定到不同类型的数据源，简化数据绑定的过程。
- 登录控件：简化创建用户登录页面的过程，使开发人员更容易编写用户授权和管理的程序。
- WebParts 控件：WebParts 是 ASP.NET 中用于构建组件化的、高度可配置的 Web 门户的一套 ASP.NET 编程控件。

- ASP.NET AJAX 控件：允许开发人员在 Web 应用程序中使用 AJAX 技术，而不需要编写大量的客户端代码。

注意：HTML 控件与 Web 控件是有区别的。HTML 控件运行在客户端，而 Web 控件是运行在服务器端的；Web 控件具有回传功能，能够使用状态保持维护控件状态；HTML 控件的事件发生后由浏览器来执行和处理，而 Web 控件的事件是由浏览器发送消息交给服务器来处理(在任何 HTML 控件中添加一个在服务器端运行的属性 runat="server"，就成了 HTML 服务器控件)。

2. 在页面中添加 HTML 服务器控件

给 HTML 标记添加 runat="server"属性，该标记就变成了 HTML 服务器控件。每个 HTML 服务器控件都是一个对象，因此，可以在服务器上以编程方式访问其属性和方法，并为其编写在服务器端运行的事件处理程序。

有如下的代码：

```
< input id = "Button1" type = "button" value = "button" />
```

添加服务器端属性之后的代码如下：

```
< input id = "Button1" type = "button" value = "button" runat = "server" />
```

比较上面两行代码，可以看出，只要在控件中添加一个 runat="server"的属性即可。

3. 在页面中添加 Web 服务器控件

添加 Web 服务器控件有两种方式：可以通过工具箱选择待添加的控件，然后直接将该控件拖动到需要添加的页面位置；也可以直接进入页面的源视图，通过 HTML 语法直接将该控件添加到页面的相应位置。

【示例 3-1】 演示 Web 服务器控件的添加。

(1) 启动 Visual Studio 2010 后，选择"文件"→"新建"→"网站"命令，将网站命名为 ch03，在网站中添加一个新的 aspx 页面，命名为 AddWebControl.aspx。

(2) 双击新建的页面，进入页面的设计视图。打开"工具箱"，在"标准"控件组中选择 TextBox 控件，然后将其拖到页面中。这时页面的设计视图中会自动出现一个 TextBox 控件，该控件的默认名称为 TextBox1.

(3) 切换到页面的源视图，可以看到，在页面中自动增加了如下代码：

```
< asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox >
```

通过上述步骤可以看出，如果要在页面中添加一个控件，通过源视图的 HTML 代码或通过设计视图的可视化编辑，均可以完成控件的添加。

4. 设置服务器控件属性

每个控件都有自己的属性，如 ID、Text 属性等，通过设置不同的属性，可以改变服务器

控件的展示内容和显示风格。

在 ASP.NET 中，可以通过三种方式来设置服务器控件的属性，分别是通过“属性”对话框直接设置；在控件的 HTML 代码中设置；或者通过页面的后台代码以编程的方式指定控件的属性。

通过“属性”窗口直接进行设置是最简单的方式。设置的时候，只需在设计视图下右击该控件，从弹出的快捷菜单中选择“属性”命令，即可对控件的属性进行设置。如图 3-2 所示。

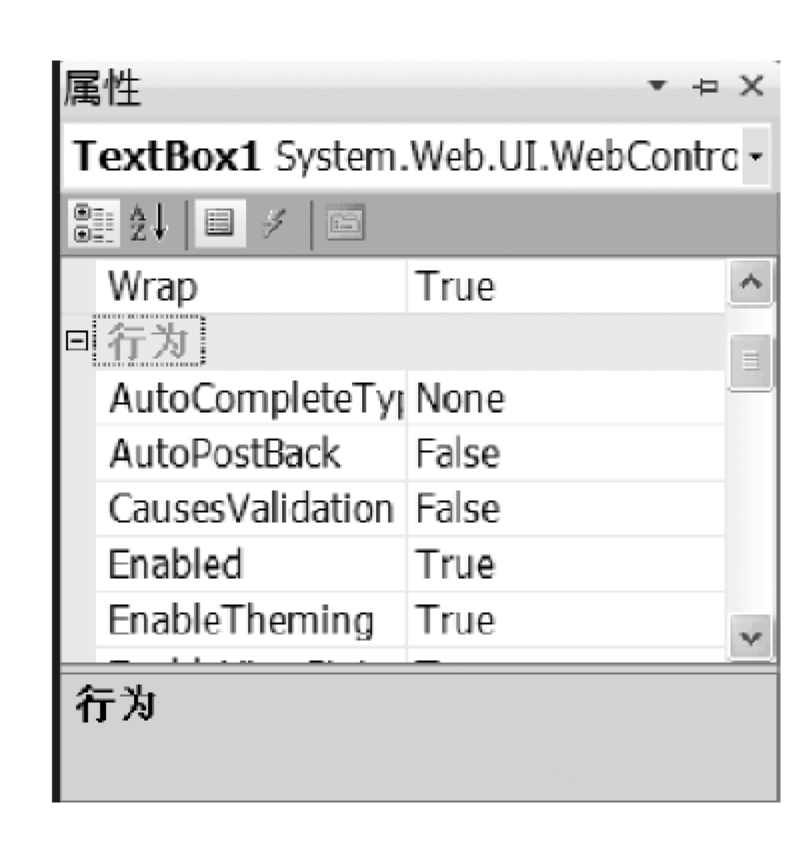


图 3-2　控件的“属性”设置窗口

通过“属性”窗口设置的控件属性，会自动更新到页面该控件的 HTML 代码中。如果对控件的某些属性比较熟悉，也可以在控件的 HTML 代码中直接编写代码，但对属性内容的设置，必须参照每个控件的声明语法，设置语法中存在的属性和值。

对控件的 HTML 代码进行设置非常方便，Visual Studio 2010 会根据控件的类型给予智能提示，即在每个控件的作用域内按空格键，会弹出该控件在此作用域内的所有可设置属性。

除了设置控件的初始属性之外，控件的属性也可以通过编程的方法在页面相应代码区域编写，设置经过某些响应或事件之后控件的属性信息，示例代码如下所示。

```
protected void Page_Load(object sender, EventArgs e)
{
  TextBox1.Text = "You are Wellcom!";     //在 Page_Load 中设置 TextBox1 的 Text 值
}
```

上述代码编写了一个 Page_Load(页面加载)事件，当页面初次被加载时，会执行 Page_Load 中的代码。这里通过编程的方式对控件的属性进行设置，当页面加载时，控件的 Text 属性值会被呈现。

3.1.2　标准服务器控件

1. 标签控件(Label)

使用 Label 控件可以在页面上的固定位置显示文本。与静态文本不同，可以通过设置 Text 属性来自定义所显示的文本，示例代码如下所示。

```
<asp:Label ID = "Label1" runat = "server" Text = "Label"></asp:Label>
```

上述代码中，声明了一个标签控件，并将这个标签控件的 ID 属性设置为默认值 Label1。由于该控件是服务器端控件，所以在控件属性中包含 runat=“server”属性。该代码还将标签控件的文本初始化为 Label，开发人员能够配置该属性进行不同文本内容的呈现。

同样,标签控件的属性能够在相应的.cs代码中初始化,示例代码如下所示。

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Label1.Text = "Hello World";          //标签赋值
}
```

注意:通常情况下,控件的ID也应该遵循良好的命名规范,以便维护。

上述代码在页面初始化时为Label1的文本属性设置为"Hello World"。值得注意的是,对于Label标签,同样也可以显示HTML样式,示例代码如下所示。

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Label1.Text = "Hello World<hr/><span style=\"color:red\">A Html Code</span>";
  //输出 HTML
    Label1.Font.Size = FontUnit.XXLarge;        //设置字体大小
}
```



图3-3 Label的Text属性的使用

上述代码中,Label1的文本属性被设置为一串HTML代码,当Label文本被呈现时,会以HTML效果显示,运行结果如图3-3所示。

如果开发人员只是为了显示一般的文本或者HTML效果,不推荐使用Label控件,因为当服务器控件过多,会导致性能问题。使用静态的HTML文本能够让页面解析速度更快。

2. 超链接控件(HyperLink)

HyperLink控件为创建超链接提供了一种简便的方法。语法格式为:

```
<asp:HyperLink id="控件名称"
Text="显示文字"
NavigateUrl="URL 地址"
Target="目标框架,默认为本框架,_blank 为新窗口"
runat="server" />
```

如果将图像文件的路径指定为ImageUrl属性,那么这个图像就会取代Text属性,成为<a>元素中的内容,例如:

```
<asp:HyperLink ID="HyperLink1" runat="server"
    NavigateUrl="http://www.microsoft.com"
    Target="_blank" Text="HyperLink"/>
```

注意:与标签控件相同的是,如果只是为了单纯的实现超链接,同样不推荐使用HyperLink控件,因为过多的使用服务器控件同样有可能造成性能问题。

3. 图像控件(Image)

图像控件用来在 Web 窗体中显示图像,图像控件常用的属性如下:

- AlternateText——在图像无法显示时显示的备用文本。
- ImageAlign——图像的对齐方式。
- ImageUrl——要显示图像的 URL。

当图片无法显示的时候,图片将被替换成 AlternateText 属性中的文字。ImageAlign 属性用来控制图片的对齐方式,而 ImageUrl 属性用来设置图像链接地址。同样,HTML 中也可以使用<img src=“ ” alt=“ ”>来替代图像控件,图像控件具有可控的优点,就是通过编程来控制图像控件。图像控件基本声明代码如下所示。

```
<asp:Image ID="Image1" runat="server" />
```

除了显示图形以外,Image 控件的其他属性还允许为图像指定各种文本,各属性如下所示。

- ToolTip——浏览器显示在工具提示中的文本。
- GenerateEmptyAlternateText——如果将此属性设置为 true,则呈现的图片的 alt 属性将设置为空。

开发人员能够为 Image 控件配置相应的属性,以便在浏览时呈现不同的样式,创建一个 Image 控件也可以直接编写 HTML 代码实现。示例代码如下所示。

```
<asp:Image ID="Image1" runat="server"
AlternateText="图片连接失效" ImageUrl="~/images/calendar.gif" />
```

上述代码设置了一个图片,并当图片失效的时候提示图片连接失效。

注意:当双击图像控件时,系统并没有生成事件所需要的代码段,这说明 Image 控件不支持任何事件。

4. TextBox(文本框)控件

通常情况下,默认的文本控件(TextBox)是一个单行的文本框,用户只能在文本框中输入一行内容。通过修改该属性,则可以将文本框设置为多行或者是以密码形式显示。TextBox 的语法格式如下:

```
<asp:Textbox id="控件名称" TextMode=" SingleLine | Multiline | Password"
Text="显示的文字" MaxLength="整数,表示输入的最大的字符数"
Rows="整数,当为多行文本时的行数" Columns="整数,当为多行文本时的列数"
Wrap="True | False,表示当控件内容超过控件宽度时是否自动换行"
AutoPostBack="True | False,表示在文本修改以后,是否自动上传数据"
OnTextChanged="当文字改变时触发的事件过程" runat="server" />
```

文本框控件常用的控件属性如下:

- AutoPostBack——在文本修改以后,是否自动重传。

- Columns——文本框的宽度。
- EnableViewState——控件是否自动保存其状态以用于往返过程。
- MaxLength——用户输入的最大字符数。
- ReadOnly——是否为只读。
- Rows——作为多行文本框时所显示的行数。
- TextMode——文本框的模式,设置单行、多行或者密码形式。
- Wrap——文本框是否自动换行。

1) AutoPostBack(自动回传)属性

在网页的交互中,如果用户提交了表单,或者执行了相应的方法,那么该页面将会发送到服务器上,服务器在执行表单的操作或者执行相应方法后,再呈现给用户,例如按钮控件、下拉菜单控件等。如果将某个控件的 AutoPostBack 属性设置为 true 时,则如果该控件的属性被修改,那么同样会使页面自动发回到服务器。

2) EnableViewState(控件状态)属性

ViewState 是 ASP.NET 中用来保存 Web 控件回传状态的一种机制,它是由 ASP.NET 页面框架管理的一个隐藏字段。在回传发生时,ViewState 数据同样将回传到服务器,ASP.NET 框架解析 ViewState 字符串并为页面中的各个控件填充该属性。而填充后,控件通过使用 ViewState 将数据重新恢复到以前的状态。

在使用某些特殊的控件(如数据库控件),来显示数据库时,每次打开页面执行一次数据库往返过程是非常不明智的。开发人员可以绑定数据,在加载页面时仅对页面设置一次,在后续的回传中,控件将自动从 ViewState 中重新填充,减少了数据库的往返次数,从而不使用过多的服务器资源。在默认情况下,EnableViewState 的属性值通常为 true。

3) 其他属性

AutoPoutBack 和 EnableViewState 属性是比较重要的属性,其他的属性也经常使用。

- MaxLength:在注册时可以限制用户输入的字符串长度。
- ReadOnly:如果将此属性设置为 true,那么文本框内的值是无法被修改的。
- TextMode:此属性可以设置文本框的模式,例如单行、多行或密码形式。默认情况下,不设置 TextMode 属性,那么文本框默认为单行。

【示例 3-2】 演示文本框控件 TextBox 的使用。

(1) 创建页面文件 TextBoxDemo.aspx,从工具箱中拖放 3 个文本框控件,或添加代码如表 3-1 所示。

表 3-1 页面 TextBoxDemo.aspx 的代码

行号	代 码 页
01	用户名:<asp:TextBox ID = "TextBox1" runat = "server"></asp:TextBox>
02	密 码:<asp:TextBox ID = "TextBox3" runat = "server" TextMode = "Password"></asp:TextBox>
03	信 息:<asp:TextBox ID = "TextBox2" runat = "server" Height = "101px" TextMode = "MultiLine"
04	Width = "325px"></asp:TextBox>

（2）浏览 TextBoxDemo.aspx 页面，结果如图 3-4 所示。

无论是在 Web 网站开发还是 WinForm 应用开发中，文本框控件都是非常重要的。文本框在用户交互中能够起到非常关键的作用。



图 3-4　文本框的 3 种形式

5．按钮控件（Button、LinkButton、ImageButton）

在 Web 应用程序和用户交互时，常常用到提交表单、获取表单信息等操作。在这其间，按钮控件是非常必要的。按钮控件能够触发事件，或者将网页中的信息回传给服务器。在 ASP.NET 中，包含 3 类按钮控件，分别为 Button、LinkButton、ImageButton。三种控件的比较如表 3-2 所示。

表 3-2　按钮控件的比较

控件	说　　明
Button	显示一个标准命令按钮，该按钮呈现为一个 HTML 的 input 元素
LinkButton	呈现为页面中的一个超链接。但是，它包含使窗体被发回服务器的客户端脚本（可以使用 HyperLink 服务器控件创建真实的超链接）
ImageButton	按钮以图形形式呈现。这对于提供丰富的按钮外观非常有用

Button（按钮）控件是一个普通按钮控件，其语法格式如下：

```
< asp:Button id = "控件名称" Text = "按钮上的文字"
CommandArgument = "此按钮管理的命令参数"
CommandName = "与此按钮关联的命令"
OnCommand = "事件过程名称"
OnClick = "事件过程名称" runat = "server"/>
```

ImageButton（图像按钮）控件用来创建一个图像提交按钮，其语法格式如下：

```
< asp:ImageButton id = "控件名称" ImageUrl = "要显示图像的 URL" OnClick = "事件过程名称"
                runat = "server" />
```

LinkButton（超链接按钮）控件是一个具有超链接的外观和普通按钮功能的控件，其语

法格式如下：

```
<asp:linkbutton id="控件名称"  Text="按钮上的文字"  OnClick="事件过程名称"
                runat="server" />
```

1）按钮事件及回传行为

当用户单击按钮控件时，该页面回传到服务器。默认情况下，将该页回传到其本身，重新生成相同页面并处理该页面上控件的事件处理程序。

可以配置按钮将当前页面回传到另一页面。这对于创建多页窗体非常有用。

按钮控件用于事件的提交，它包含一些通用属性。按钮控件的常用通用属性有：

- CausesValidation——按钮是否导致激发验证检查。
- CommandArgument——与此按钮关联的命令参数。
- CommandName——与此按钮关联的命令。
- ValidationGroup——使用该属性可以指定单击按钮时调用页面上的哪些验证程序。如果未建立任何验证组，则会调用页面上的所有验证程序。

这3种按钮控件对应的事件通常是Click（单击）和Command（命令）事件，在Click事件中，通常用于编写用户单击按钮时所需要执行的事件。

【示例3-3】 演示Button、LinkButton、ImageButton控件的Click事件。

（1）创建页面文件ThreeButtonDemo.aspx，从工具箱中分别拖放Button，LinkButton，ImageButton控件及3个Label控件至页面并完善代码如表3-3所示。

表3-3 ThreeButtonDemo.aspx页面主体部分的代码

行号	代 码 页
01	<body>
02	<form id="form1" runat="server">
03	<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Button" />
04	普通的按钮
05	
06	<asp:LinkButton ID="LinkButton1" runat="server"
07	OnClick="LinkButton1_Click">LinkButton</asp:LinkButton>
08	Link类型的按钮
09	
10	<asp:ImageButton ID="ImageButton1" runat="server" ImageUrl="~/images/image.png"
11	Height="50" AlternateText="this is a ImageButton." OnClick="ImageButton1_Click" />
12	图像类型的按钮
13	
14	<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
15	
16	<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>

续表

代　码　页

```
            <br />
            <asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
        </form>
    </body>
```

（2）在 ThreeButtonDemo.aspx.cs 中添加代码如表 3-4 所示。

表 3-4　页面 ThreeButtonDemo.aspx 对应 cs 文件的代码

代　码　页

```
protected void Button1_Click(object sender, EventArgs e)
{
    Label1.Text = "普通按钮被触发";      //输出信息
}
protected void LinkButton1_Click(object sender, EventArgs e)
{
    Label2.Text = "超链接按钮被触发";    //输出信息
}
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    Label3.Text = "图片按钮被触发";    //输出信息
}
```

表 3-4 中代码分别为 3 种按钮生成了事件，将 Label1、Label2、Label3 的文本设置为相应文本。

（3）运行后，分别单击 Button、LinkButton 和图片，效果如图 3-5 所示。

2）Command 事件

按钮控件中的 Click 事件并不能传递参数，所以处理的事件相对简单。而 Command 事件可以传递参数，负责传递参数的是按钮控件的 CommandArgument 和 CommandName 属性，如图 3-6 所示。



图 3-5　3 种类型按钮的 Click 事件触发后的效果



图 3-6　CommandArgument 和 CommandName 属性

将示例 3-3 中按钮 Button1 的 CommandArgument 和 CommandName 属性分别设置为 Hello 和 FirstBtn，单击创建一个 Command 事件并在事件中编写相应代码，示例代码如下所示。

```
protected void Button1_Command(object sender, CommandEventArgs e)
{
    //如果 CommandName 属性的值为 FirstBtn，则运行下面代码
    if (e.CommandName == "FirstBtn")
    {
        Label1.Text = e.CommandArgument.ToString();
        //CommandArgument 属性的值赋给 Label1
    }
}
```

注意：当按钮同时包含 Click 和 Command 事件时，通常情况下会执行 Command 事件。Command 有一些 Click 不具备的好处，就是传递参数。可以对按钮的 CommandArgument 和 CommandName 属性分别设置，通过判断 CommandArgument 和 CommandName 属性来执行相应的方法。这样，一个按钮控件就能够实现不同的方法，使得多个按钮与一个处理代码关联或者一个按钮根据不同的值进行不同的处理和响应。相比 Click 事件而言，Command 事件具有更高的可控性。

6. 单选控件和单选组控件（RadioButton 和 RadioButtonList）

1）单选控件（RadioButton）

单选控件可以为用户提供单选选项，单选控件常用属性如下所示。

- Checked：控件是否被选中。
- GroupName：单选控件所处的组名。
- TextAlign：文本标签相对于控件的对齐方式。

单选控件通常需要 Checked 属性来判断某个选项是否被选中，多个单选控件之间可能存在着某些联系，这些联系通过 GroupName 进行约束和联系，示例代码如下所示。

```
<asp:RadioButton ID = "RadioButton1" runat = "server" GroupName = "choose" Text = "男"
 AutoPostBack = "True"  OnCheckedChanged = "RadioButton1_CheckedChanged" />
<asp:RadioButton ID = "RadioButton2" runat = "server" GroupName = "choose" Text = "女"
 AutoPostBack = "True"  OnCheckedChanged = "RadioButton2_CheckedChanged" />
```

上述代码声明了两个单选控件，并将 GroupName 属性都设置为 choose。单选控件中最常用的事件是 CheckedChanged，当控件的选中状态改变时，则触发该事件，示例代码如下所示。

```
protected void RadioButton1_CheckedChanged(object sender, EventArgs e)
{
    Label1.Text = "你选择的是" + RadioButton1.Text;
}
```

```
protected void RadioButton2_CheckedChanged(object sender, EventArgs e)
{
    Label1.Text = "你选择的是" + RadioButton2.Text;
}
```

上述代码中，当选中状态被改变时，则触发相应的事件，运行结果如图 3-7 所示。

注意：与 TextBox 文本框控件相同的是，单选控件不会自动进行页面回传，必须将 AutoPostBack 属性设置为 true 时才能在焦点丢失时触发相应的 CheckedChanged 事件。

2）单选组控件(RadioButtonList)

与单选控件相同，单选组控件也是只能选择一个项目的控件，而与单选控件不同的是，单选组控件没有 GroupName 属性，但是却能够列出多个单选项目。另外，单选组控件所生成的代码也比单选控件实现的相对较少。示例代码如下所示：



图 3-7 单选控件的使用

```
<asp:RadioButtonList ID="RadioButtonList1" runat="server" RepeatDirection="Horizontal">
  <asp:ListItem Selected="True">男</asp:ListItem>
  <asp:ListItem>女</asp:ListItem>
</asp:RadioButtonList>
```

7. CheckBox(复选框)和 CheckBoxList(复选框列表)控件

CheckBox 控件和 CheckBoxList 控件分别用于向用户提供选项和选项列表。CheckBox 控件适合用在选项不多且比较固定的情况，CheckBoxList 控件适合选项较多或者需要在运行时动态决定有哪些选项时较为方便。CheckBox 控件的语法格式如下：

```
<asp: CheckBox  ID="控件名"  runat="server"  Text="控件的文字"  value=""/>
```

CheckBox 控件的常用属性如下：

- Checked 属性——获取或设置该项是否选中。
- TextAlign 属性——控件文字的位置。
- Text 属性——获取或设置 CheckBox 控件的文本内容。
- Value 属性——获取或设置 CheckBox 控件的值内容。
- AutoPostBack 属性——获取或设置当改变 CheckBox 控件的选择状态时，是否自动回传窗体数据到服务器。值为 True 时，表示单击 CheckBox 控件，页面自动回传；值为 False 时，不回传。默认值为 False。
- CheckBox 控件具有 CheckedChanged 事件。当 Checked 属性的值改变时，会触发此事件。与 TextBox 控件类似，该事件要与 AutoPostBack 属性配合使用。

CheckBoxList 控件的语法格式如下：

```
<asp: CheckBoxList  ID="控件名"  Runat="server">
    <asp: ListItem  value="" > text </asp: ListItem>
      ⋮
</asp: CheckedBoxList>
```

该控件的属性、用法及功能与 ListBox 控件基本相同。除此之外，还有自己的特殊属性。

- RepeatDirection：表示是横向还是纵向排列(Vertical|Horizontal)。
- RepeatColumns：一行排几列。
- TextAlign 属性：控件文字的位置。
- Selected 属性：表示该选项是否选中。

【示例 3-4】 演示 CheckBox、CheckBoxList 的使用。

(1) 创建页面文件 CheckBox_CheckBoxList.aspx，从工具箱中拖放 1 个 Button、1 个 Label、2 个 CheckBox 和 1 个 CheckBoxList 控件。

(2) 在设置 CheckBoxList 的选项时，可以通过 ListItem 窗口进行设置。在页面设计视图下选中 CheckBoxList 控件，在 CheckBoxList 任务下选择“编辑项”选项来打开“ListItem 集合编辑器”对话框，单击“添加”按钮，可以添加多选项，如图 3-8 所示。

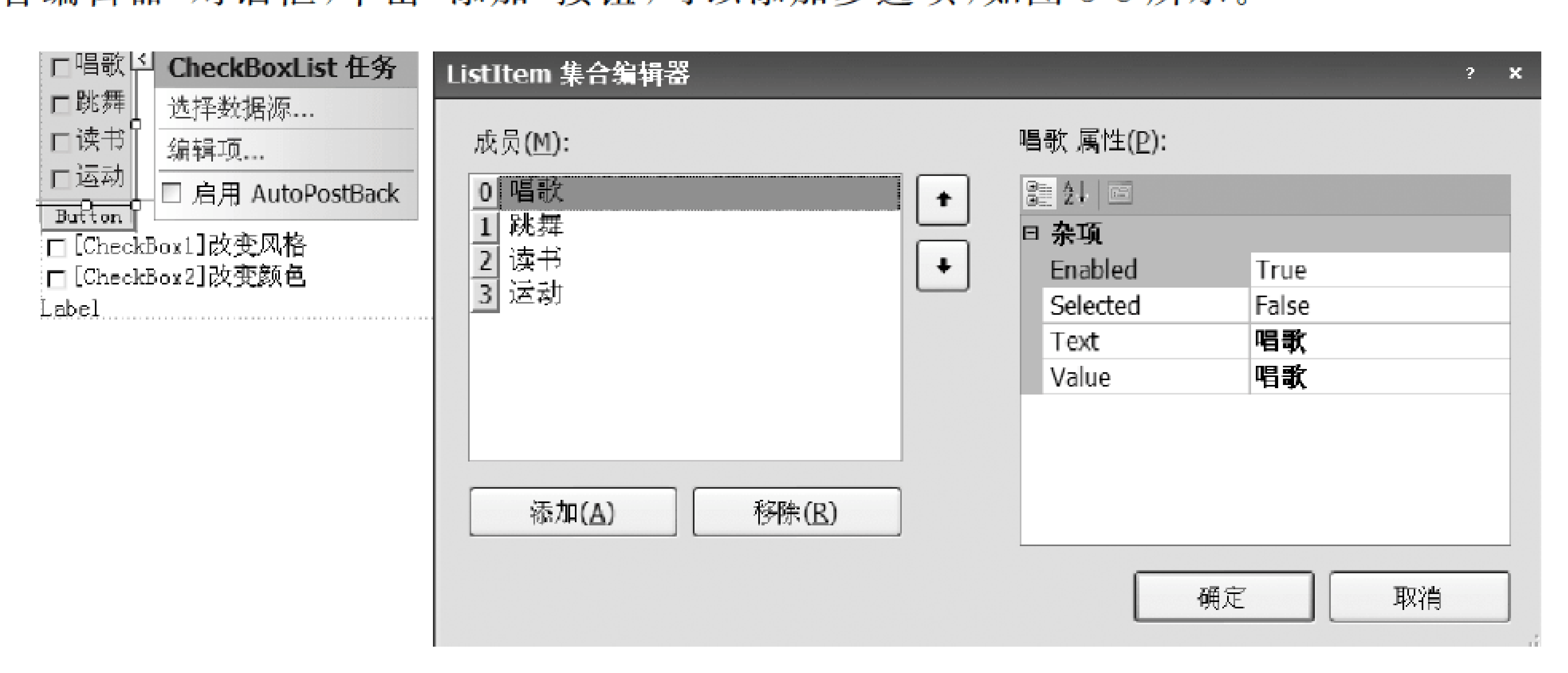


图 3-8 “ListItem 集合编辑器”对话框

(3) 根据需要在页面 CheckBox_CheckBoxList.aspx 添加相应代码，如表 3-5 所示。

表 3-5 页面 CheckBox_CheckBoxList.aspx 主体部分的代码

行号	代 码 页
01	<body>
02	<form id="form1" runat="server">
03	<asp:CheckBoxList ID="CheckBoxList1" runat="server">
04	<asp:ListItem>唱歌</asp:ListItem>
05	<asp:ListItem>跳舞</asp:ListItem>
06	<asp:ListItem>读书</asp:ListItem>
07	<asp:ListItem>运动</asp:ListItem>
08	</asp:CheckBoxList>
09	<asp:Button ID="Button1" runat="server" OnClick="Button1_Click1" Text="Button" />

续表

```
        <br />
        <asp:CheckBox ID="CheckBox1" runat="server"
              OnCheckedChanged="CheckBox1_CheckedChanged1" />改变风格
        <br />
        <asp:CheckBox ID="CheckBox2" runat="server"
              OnCheckedChanged="CheckBox2_CheckedChanged1" />改变颜色
        <br />
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        <br />
        </form>
    </body>
```

（4）在 CheckBox_CheckBoxList.aspx.cs 中添加代码，如表 3-6 所示。

表 3-6　页面 CheckBox_CheckBoxList.aspx 对应 cs 文件的代码

```
…
using System.Drawing;
public partial class ch3_4_CheckBox_CheckBoxList : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Button1_Click1(object sender, EventArgs e)
    {
        string str = "选择结果：";
        Label1.Text = "";
        for (int i = 0; i < CheckBoxList1.Items.Count; i++)
        {
            if (CheckBoxList1.Items[i].Selected)
            {
                str += CheckBoxList1.Items[i].Text + "、";
            }
        }
        if (str.EndsWith("、") == true) str = str.Substring(0, str.Length - 1);
        Label1.Text = str;
        if (str == "选择结果：")
        {
            string scriptString = "alert('请作出选择!');";
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "warning!", scriptString, true);
        }
        else
```

续表

```
            {
                Label1.Visible = true;
                Label1.Text = str;
            }
        }
        protected void CheckBox1_CheckedChanged1(object sender, EventArgs e)
        {
            this.CheckBoxList1.BackColor = CheckBox1.Checked ? Color.Beige : Color.Azure;
            CheckBoxList1.RepeatDirection =
              CheckBox1.Checked ? RepeatDirection.Horizontal : RepeatDirection.Vertical;
        }
        protected void CheckBox2_CheckedChanged1(object sender, EventArgs e)
        {
            if (CheckBox2.Checked)
            {
                this.CheckBoxList1.ForeColor = Color.Red;
                Label1.ForeColor = Color.Red;
            }
            else
            {
                this.CheckBoxList1.ForeColor = Color.Black;
                Label1.ForeColor = Color.Black;
            }
        }
    }
```

(5) 运行页面,效果如图 3-9 和图 3-10 所示。



图 3-9　运行初始状态



图 3-10　选择复选框后单击 Button

8. 列表控件(DropDownList 和 ListBox)

在 Web 开发中,经常会需要使用列表控件,让用户的输入更加简单。例如在用户注册时,用户的所在地是有限的集合,而且用户不喜欢经常输入,这样就可以使用列表控件。同样列表控件还能够简化用户输入并且防止用户输入在实际中不存在的数据。

1) DropDownList 列表控件

列表控件能在一个控件中为用户提供多个选项,同时又能够避免用户输入错误的选项。DropDownList 是一个单项选择下拉列表框控件,其语法格式如下:

```
<asp:DropDownList  ID="控件名"  runat="server" >
    <asp:ListItem Value="" >Text</asp:ListItem>
    ⋮
</asp:DropDownList>
```

DropDownList 控件的主要属性如下:

- AutoPostBack 属性——获取或设置当改变 DropDownList 控件的选择状态时,是否自动上传窗体数据到服务器。默认为 False。
- Items 属性——包含该控件所有选项的集合。每个列表项都是一个单独的对象,具有自己的属性。
- SelectedIndex 属性——获取当前选择项的下标(下标从 0 开始)。
- SelectedItem 属性——获取当前选择项对象。

DropDownList 控件有 SelectedIndexChanged 事件,当用户选择一项时,DropDownList 控件将引发 SelectedIndexChanged 事件。

默认情况下,此事件不会导致向服务器发送页面,但当该控件的 AutoPostBack 属性设置为 True 时,该事件会立即回传页面。

2) ListBox 列表控件

ListBox 控件与 DropDownList 控件的功能基本相似,ListBox 控件将所有选项都显示出来,提供单选或多选的列表框。

ListBox 控件比 DropDownList 控件多两个属性:

- Rows 属性——获取或设置 ListBox 控件显示的选项行数,默认值为 4。
- SelectionMode 属性——获取或设置 ListBox 控件的选项模式,Single 为单选,Multiple 为多选,默认为 Single。当允许多选时,只需按住 Ctrl 键或 Shift 键并单击要选取的选项,便可完成多选。

【示例 3-5】 演示 DropDownList、ListBox 控件的使用。

(1) 创建页面文件 DropDownList.aspx,从工具箱中拖放 1 个 DropDownList 和 2 个 Label 控件,通过 ListItem 为 DropDownList 和 ListBox 控件添加选项,代码如表 3-7 所示。

(2) 在 DropDownList.aspx.cs 中添加代码如表 3-8 所示。

表 3-7 页面 DropDownList.aspx 主体部分的代码

```
<body>
    <form id="form1" runat="server">
    <div>
        <asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="True"
          OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged1">
          <asp:ListItem>请选择一门课程</asp:ListItem>
          <asp:ListItem>ASP.NET</asp:ListItem>
          <asp:ListItem>Java</asp:ListItem>
          <asp:ListItem>英语</asp:ListItem>
          <asp:ListItem>数据结构</asp:ListItem>
          <asp:ListItem>操作系统</asp:ListItem>
          <asp:ListItem>数据库原理</asp:ListItem>
        </asp:DropDownList>
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        <br />
        <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
          OnSelectedIndexChanged="ListBox1_SelectedIndexChanged"
          SelectionMode="Multiple">
          <asp:ListItem>请选择多门课程</asp:ListItem>
          <asp:ListItem>ASP.NET</asp:ListItem>
          <asp:ListItem>Java</asp:ListItem>
          <asp:ListItem>英语</asp:ListItem>
          <asp:ListItem>数据结构</asp:ListItem>
          <asp:ListItem>操作系统</asp:ListItem>
          <asp:ListItem>数据库原理</asp:ListItem>
        </asp:ListBox>
        <asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
    </div>
    </form>
</body>
```

表 3-8 页面 DropDownList.aspx 对应的 cs 文件的代码

```
protected void DropDownList1_SelectedIndexChanged1(object sender, EventArgs e)
{
    Label1.Text = "你选择了" + DropDownList1.Text + "课程";
}
protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    string str = "";
    Label2.Text = "";
```

续表

```
        for (int i = 0; i < ListBox1.Items.Count; i++)
        {
            if (ListBox1.Items[i].Selected)
            {
                str += ListBox1.Items[i].Text + "、";
            }
        }
        Label2.Text = "你选择了" + str + "课程";
    }
```

(3) 运行页面,效果如图 3-11 和图 3-12 所示。



图 3-11　列表初始状态



图 3-12　列表选择之后的效果

注意:ListBox 控件中如果允许用户选择多项,需要设置其 SelectionMode 属性为 Multiple。

3.1.3　验证控件

网页交互过程中,经常需要使用输入控件来收集用户输入的信息。为确保用户提交到服务器的信息在内容和格式上都是合法的,就必须编写代码来验证用户输入的内容。ASP.NET 提供了强大的验证控件,它可以验证服务器控件中用户的输入,并在验证失败的情况下显示一条自定义错误消息。验证控件直接在客户端执行,用户提交后执行相应的验证无需使用服务器端进行验证操作,从而减少了服务器与客户端之间的往返过程。

1. 验证控件及其作用

ASP.NET 验证控件是一个服务器控件集合,允许这些控件验证关联的输入服务器控件(如 TextBox),并在验证失败时显示自定义消息。每个验证控件执行特定类型的验证,一个输入控件可以同时被多个验证控件关联验证。ASP.NET 的验证控件如表 3-9 所示。

表 3-9　ASP.NET 的验证控件

验证类型	使用的控件	说　明
必需项	RequiredFieldValidator	验证一个必填字段,确保用户不会跳过该项输入
与某值的比较	CompareValidator	将用户输入与一个常数值或者另一个控件或特定数据类型的值进行比较(使用小于、等于或大于比较运算符)
范围检查	RangeValidator	用于检查用户的输入是否在指定的上下限内。可以检查数字对、字母对和日期对的限定范围
模式匹配	RegularExpressionValidator	用于检查输入的内容与正则表达式所定义的模式是否匹配。此类验证可用于检查可预测的字符序列,例如电子邮件地址、电话号码、邮政编码等内容中的字符序列
用户定义	CustomValidator	使用自己编写的验证逻辑检查用户输入。此类验证能够检查在运行时派生的值
验证总汇	ValidationSummary	该控件不执行验证,但经常与其他验证控件一起用于显示来自网页上所有验证控件的错误信息

因此,可以通过 CompareValidator 和 RangeValidator 控件分别检查某个特定值或值范围,还可以调用 CustomValidator 控件定义自己的验证条件,或者使用 ValidationSummary 控件显示网页上所有验证控件的结果摘要。

2. 验证控件的属性和方法

所有的验证控件都继承自 BaseValidator 类,BaseValidator 类为所有的验证控件提供了一些公用的属性和方法,如表 3-10 所示。

表 3-10　验证控件的公共属性和方法

属　性	说　明
Display 属性	获取或设置验证控件中错误信息的显示行为
ErrorMessage 属性	获取或设置验证失败时 ValidationSummary 控件中显示的错误信息的文本
Text 属性	获取或设置验证失败时验证控件中显示的文本
ControlToValidate 属性	获取或设置要验证的输入控件
EnableClientScript 属性	获取或设置一个值,该值指示是否启用客户端验证
SetFocusOnError 属性	获取或设置一个值,该值指示在验证失败时是否将焦点设置到 ControlToValidate 属性指定的控件上
ValidationGroup 属性	获取或设置此验证控件所属的验证组的名称
IsValid 属性	获取或设置一个值,该值指示关联的输入控件是否通过验证
ForeColor 属性	指定当验证失败时用于显示错误消息的文本颜色

验证控件总是在服务器上执行验证检查。它们还具有完整的客户端实现,该实现允许支持 DHTML 的浏览器在客户端执行验证。客户端验证通过在向服务器发送用户输入前检查用户输入来增强验证过程。在提交窗体前即可在客户端检测到错误,从而避免了服务器端验证所需信息的来回传递。

客户端的验证经常被使用,因为它有非常快的响应速度。可以通过将 EnableClientScript 属性设置为 False 关闭客户端验证。

每个验证控件以及 Page 对象本身,都有一个 IsValid 属性,利用该属性可以进行页面

有效性的验证。只有当页面的所有验证都成功时，Page.IsValid 属性才为真。

3. 表单验证控件(RequiredFieldValidator)

在实际的应用中，如在用户填写表单时，有一些项目是必填项，例如用户名和密码。在传统的 ASP 中，当用户填写表单后，页面需要被发送到服务器并判断表单中的某项 HTML 控件的值是否为空，如果为空，则返回错误信息。在 ASP.NET 中，系统提供了 RequiredFieldValidator 验证控件进行验证。使用 RequiredFieldValidator 控件能够指定某个用户在特定的控件中必须提供相应的信息，如果不填写相应的信息。RequiredFieldValidator 控件就会提示错误信息。RequiredFieldValidator 控件示例代码如下所示：

```
<form id="form1" runat="server">
<div>
    姓名:<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
          ControlToValidate="TextBox1"  ErrorMessage="姓名不能为空!">
    </asp:RequiredFieldValidator><br />
    密码:<asp:TextBox ID="TextBox2" runat="server"></asp:TextBox><br />
    <asp:Button ID="Button1" runat="server" Text="登录" /><br />
</div>
</form>
```

在进行验证时，RequiredFieldValidator 控件必须绑定一个服务器控件。在上述代码中，验证控件 RequiredFieldValidator 的服务器控件绑定为 TextBox1，当 TextBox1 中的值为空时，则会提示自定义错误信息“姓名不能为空!”。TextBox2 没有绑定，所以没有提示，效果如图 3-13 所示。

当姓名选项未填写时，会提示必填字段不能为空，并且该验证在客户端执行。当发生此错误时，用户会立即看到该错误提示而不会立即进行页面提交，当用户填写完成并再次单击按钮控件时，页面才会向服务器提交。

图 3-13　RequiredFieldValidator 验证控件

4. 比较验证控件(CompareValidator)

比较验证控件对照特定的数据类型来验证用户的输入。因为当用户输入用户信息时，难免会输入错误信息，如需要了解用户的生日时，用户很可能输入了其他的字符串。CompareValidator 比较验证控件能够比较控件中的值是否符合开发人员的需要。CompareValidator 控件的特有属性如下：

- ControlToValidate——要验证的控件 ID。
- ControlToCompare——用来与要验证的控件进行比较的控件的 ID。
- Operator——要使用的比较运算符，比如>、>=、<=、<，默认为等于(=)。
- Type——要比较的两个值的数据类型，不同类型的比较可能会出错。

• ValueToCompare——以字符串形式输入的表达式,即用于比较的值。

当使用 CompareValidator 控件时,可以方便地判断用户是否正确输入,示例代码如下所示。

```
<form id = "form1" runat = "server">
<div>
    密码: <asp:TextBox ID = "txtPassword"  runat = "server"></asp:TextBox><br />
    重复密码: <asp:TextBox ID = "txtRePassword" runat = "server"></asp:TextBox>
    <asp:CompareValidator ID = "CompareValidator1" runat = "server"
          ControlToCompare = "txtPassword"
          ControlToValidate = "txtRePassword" Display = "Dynamic"
          ErrorMessage = "密码输入不一致">
    </asp:CompareValidator><br />
    出生年月: <asp:TextBox ID = "txtDate" runat = "server"></asp:TextBox>
    <asp:CompareValidator ID = "CompareValidator2" runat = "server"
        ControlToValidate = "txtDate"
        Display = "Dynamic" ErrorMessage = "日期格式不正确"
        Operator = "DataTypeCheck" Type = "Date">
    </asp:CompareValidator><br />
    <asp:Button ID = "btnSubmit" runat = "server" Text = "提交" /><br />
</div>
</form>
```

上述代码判断两个输入密码的文本框 txtPassword、txtRePassword 的输入的值是否相等;判断文本框 txtDate 输入的格式是否符合正确的日期格式,当输入的格式错误时,会提示错误。效果如图 3-14 所示。



图 3-14 CompareValidator 验证控件

5. 范围验证控件(RangeValidator)

范围验证控件(RangeValidator)可以检查用户的输入是否在指定的上限与下限之间。通常情况下用于检查数字、日期、货币等。范围验证控件(RangeValidator)控件的常用属性如下所示。

• MinimumValue——指定有效范围的最小值。
• MaximumValue——指定有效范围的最大值。

• Type——指定要比较的值的数据类型。

通常情况下，为了控制用户输入的范围，可以使用该控件。当输入用户的生日时，今年是 2014 年，那么用户就不应该输入 2015 年；同样，基本上没有人的寿命会超过 150，所以对输入的日期的下限也需要进行规定。示例代码如下所示。

```
<form id = "form1" runat = "server">
<div>
    年龄：<asp:TextBox ID = "txtAge" runat = "server"></asp:TextBox>
    <asp:RangeValidator ID = "RangeValidator1" runat = "server" ControlToValidate = "txtAge"
        Display = "Dynamic"  ErrorMessage = "年龄应在 0 - 200 之间"
        MaximumValue = "200"
        MinimumValue = "0"
        Type = "Integer"></asp:RangeValidator>
    <asp:Button ID = "btnSubmit" runat = "server" Text = "提交" />
</div>
</form>
```

上述代码将 MinimumValue 属性值设置为 0，并将 MaximumValue 的值设置为 200。当用户的日期低于最小值或高于最大值时，则提示错误，如图 3-15 所示。



图 3-15　CompareValidator 验证控件

注意：RangeValidator 验证控件在进行控件的值的范围设定时，其范围不仅仅可以是一个整数值，同样还可以是时间、日期等值。

6．正则验证控件（RegularExpressionValidator）

在上述控件中，虽然能够实现一些验证，但是验证的能力是有限的。例如在验证的过程中，只能验证是否是数字，或者是否是日期；也可能在验证时，只能验证一定范围内的数值。虽然这些控件提供了一些验证功能，但限制了开发人员进行自定义验证和错误信息的开发。为实现一个验证，很可能需要多个控件同时搭配使用。

正则验证控件（RegularExpressionValidator）就解决了这个问题。正则验证控件的功能非常的强大，它用于确定输入的控件的值是否与某个正则表达式所定义的模式相匹配，如电子邮件、电话号码以及序列号等。其语法格式如下：

```
<asp:RegularExpressionValidator id = "控件名称"
    ControlToValidate = "被验证的控件的名称"
```

```
ValidationExpression = "正则表达式"
ErrorMessage = "错误发生时的提示信息"
Display = "Dynamic | Static | None"
runat = "server" />
```

正则验证控件(RegularExpressionValidator)常用的属性是 ValidationExpression,它用来指定用于验证的输入控件的正则表达式。常用的正则表达式字符及其含义如表 3-11 所示。

表 3-11 常用正则表达式字符及其含义

正则表达式字符	含　义
[...]	匹配括号中的任何一个字符
[...]	匹配不在括号中的任何一个字符
\w	匹配任何一个字符(a~z、A~Z 和 0~9)
\W	匹配任何一个空白字符
\s	匹配任何一个非空白字符
\S	与任何非单词字符匹配
\d	匹配任何一个数字(0~9)
\D	匹配任何一个非数字(^0~9)
[\b]	匹配一个退格键字母
{n,m}	最少匹配前面表达式 *n* 次,最大为 *m* 次
{n,}	最少匹配前面表达式 *n* 次
{n}	恰好匹配前面表达式 *n* 次
?	匹配前面表达式 0 或 1 次{0,1}
+	至少匹配前面表达式 1 次
*	至少匹配前面表达式 0 次{0,}
|	匹配前面表达式或后面表达式
(...)	在单元中组合项目
^	匹配字符串的开头
$	匹配字符串的结尾
\b	匹配字符边界
\B	匹配非字符边界的某个位置

下面列举几个常用的正则表达式。

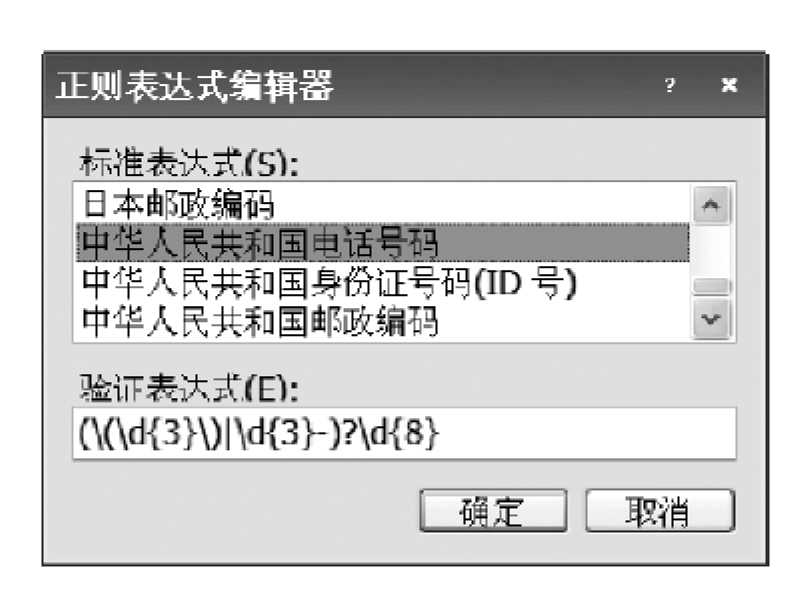
- 电话验证:[0-9]{3,4}-[0-9]{7,8},如 0371-92345678 或 010-12345678。
- 18 位身份证验证:[0-9]{6}[12][0-9]{3}[01][0-9][0123][0-9][0-9]{3}[12]。
- E-mail 验证:.{1,}@.{1,}\.[a-zA-Z]{2,3}。
- HTML 标记:<(\S*?)[^>]*>.*?</\1>|<.*?/>。
- 网址 URL:[a-zA-z]+://[^\s]*。
- 中国邮政编码:[1-9]\d{5}(?!\d)。
- IP 地址:\d+\.\d+\.\d+\.\d+。

客户端的正则表达式验证语法和服务端的正则表达式验证语法不同，因为在客户端使用的是 JScript 正则表达式语法，而在服务器端使用的是 Regex 类提供的正则表达式语法。使用正则表达式能够实现字符串的匹配并验证用户的输入的格式是否正确，系统提供了一些常用的正则表达式，开发人员能够选择相应的选项进行规则筛选。切换到页面“设计”视图，从“工具箱”→“验证”选项组中，将 RegularExpressionValidator 控件拖动到页面上，选择此控件，然后在“属性”窗口中找到“行为”下的 ValidationExpression 属性，单击 ValidationExpression 属性右边的省略符号按钮，即可打开“正则表达式编辑器”对话框，如图 3-16 所示。

当选择了正则表达式后，系统自动生成的 HTML 代码如下所示。

```
<form id="form1" runat="server">
    Telphone: <asp:TextBox ID="txtTel" runat="server" Height="22px"></asp:TextBox>
    <asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server"
         ControlToValidate="txtTel"
         ErrorMessage="请输入合法的电话号码"
         ValidationExpression="[0-9]{3,4}-[0-9]{7,8}"></asp:RegularExpressionValidator>
    <br />
    <asp:Button ID="btnSubmit" runat="server" Text="提交" /></div>
</form>
```

运行后当用户单击按钮控件时，如果输入的信息与相应的正则表达式不匹配，则会提示错误信息，如图 3-17 所示。



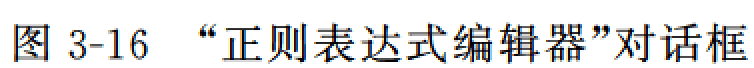

图 3-16　“正则表达式编辑器”对话框



图 3-17　RegularExpressionValidator 验证控件

同样，开发人员也可以自定义正则表达式来规范用户的输入。使用正则表达式能够加快验证速度并在字符串中快速匹配；而另一方面，使用正则表达式能够减少复杂的应用程序的功能开发和实现。

注意：在用户输入为空时，其他的验证控件都会验证通过。所以，验证控件通常需要同表单验证控件(RequiredFieldValidator)一起使用。

7. 验证组控件(ValidationSummary)

验证组控件(ValidationSummary)本身没有验证功能，但验证组控件(ValidationSummary)通过 ErrorMessage 属性为页面上的每个验证控件显示错误信息。验证组控件(ValidationSummary)的常用属性如下：

- DisplayMode——摘要可显示为列表、项目符号列表或单个段落。
- HeaderText——标题部分指定一个自定义标题。
- ShowMessageBox——是否在消息框中显示摘要。
- ShowSummary——控制是显示还是隐藏 ValidationSummary 控件。

验证控件能够显示页面的多个控件产生的错误,示例代码如下所示。

```
<form id="form1" runat="server">
<div>
    姓名:
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
          ErrorMessage="姓名为必填项"
          ControlToValidate="TextBox1">
    </asp:RequiredFieldValidator>
    <br />
    身份证:
    <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
    <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
          ControlToValidate="TextBox2"
          ErrorMessage="身份证号码错误"
          ValidationExpression="\d{17}[\d|X]|\d{15}">
    </asp:RegularExpressionValidator>
    <br />
    <asp:Button ID="Button1" runat="server" Text="提交" />
    <asp:ValidationSummary ID="ValidationSummary1" runat="server" />
</div>
</form>
```

上述代码运行效果如图 3-18 所示



图 3-18 RegularExpressionValidator 验证控件

当有多个错误发生时,ValidationSummary 控件能够捕获多个验证错误并呈现给用户,这样就避免了一个表单需要多个验证时需要使用多个验证控件进行绑定,使用 ValidationSummary 控件就无须为每个需要验证的控件进行绑定。

3.1.4 FileUpload 控件

1. FileUpload 控件概述

FileUpload(文件上传)控件主要功能是上传文件到服务器。该控件提供一个文本框和一个“浏览”按钮,用户可以在文本框中输入完整的文件路径,或者单击“浏览”按钮从客户端选择需要上传的文件,然后在服务器中调用 SaveAs 方法保存上传的文件,也可以通过 FileContent 属性获取需要上传的 Stream 对象。通常把 Stream 对象保存到数据库。FileUpload 控件不会自动上传文件,必须设置相关的事件处理程序,并在程序中实现文件上传。

FileUpload 控件提供了一些属性、方法和事件实现文件上传功能,如表 3-12 所示。

表 3-12 FileUpload 控件常用属性、方法

属性和方法	说明
FileBytes 属性	获取上传文件的字节数组
FileContent 属性	获取上传文件的文件流(Stream)对象
FileName 属性	获取上传文件在客户端的文件名称
HasFile 属性	确定是否有上传文件,表示 FileUpload 控件是否已包含一个文件
PostedFile 属性	获取一个与上传文件相关的 HttpPostedFile 对象,获取相关属性
SaveAs 方法	将上传的文件保存到指定的路径

提示:使用 FileUpload 控件,一般要导入命名空间 System.IO,用于在服务器端操作文件目录。

2. FileUpload 控件应用

【示例 3-6】 通过 FileUpload 控件上传图片文件,并将源文件的路径、文件大小和文件类型显示出来。

(1) 创建页面文件 FileUploadDemo.aspx,从工具箱中拖放一个 FileUpload 控件、一个 Label 控件和一个 Button 控件到页面中,添加一个名字为 images 的文件夹用于存放上传的文件。FileUploadDemo.aspx 页面代码如下。

```
<form id="form1" runat="server">
    <div>
        <asp:FileUpload ID="FileUpload1" runat="server" />
        <asp:Button ID="btnUpload" runat="server" OnClick="btnUpload_Click" Text="上传" />
        <asp:Label ID="lbInfo" runat="server" Text="Label"></asp:Label>
    </div>
</form>
```

(2) 在页面后置代码文件 FileUploadDemo.aspx.cs 中添加单击 Button 按钮事件的代码,如表 3-13 所示。

表 3-13 "上传"按钮的 Click 事件方法代码

```
/// <summary>
/// "上传"按钮的 Click 事件方法
/// </summary>
/// <param name = "sender"></param>
/// <param name = "e"></param>
protected void btnUpload_Click(object sender, EventArgs e)
{
    if (this.FileUpload1.HasFile == true)
    {
        if (FileUpload1.FileName == "" || FileUpload1.FileName == null)
        {
            return;
        }
        string File_N = FileUpload1.FileName.ToString();     //获取上传文件的名称
        //string webDir = Server.MapPath(".") + "\\images\\";
        string webDir = Server.MapPath("~/images/");
        if (!Directory.Exists(webDir))                        //检查目录是否存在
        {
            Directory.CreateDirectory(webDir);                //不存在,则创建
        }
        FileUpload1.SaveAs(webDir + File_N);
        this.lbInfo.Text = "<li>" + "原文件路径: " + this.FileUpload1.PostedFile.FileName;
        this.lbInfo.Text += "<br>";
        this.lbInfo.Text += "<li>" + "文件大小: " + this.FileUpload1.PostedFile.ContentLength + "字节";
        this.lbInfo.Text += "<br>";
        this.lbInfo.Text += "<li>" + "文件类型: " + this.FileUpload1.PostedFile.ContentType;
        Response.Write("文件上传成功");
    }
}
```

(3) 运行页面 FileUploadDemo.aspx,效果如图 3-19 所示。

文件上传成功后,在网站项目 ch03 下可以看到 images 文件夹下有已上传的文件(若没有 images 文件夹,则选择"刷新文件夹"选项)。

3.1.5 第三方控件

ASP.NET 4.0 虽然提供了八十多种内置控件,但都是基于自身需求而来,有它的商业目的。人们对自己开发中遇到的内容比较熟悉,于是开发出适合自己使用的控件,后来发现许多人也有这样需求,于是就放在网络上发展成"第三方控件"。下面介绍一些常用的第三

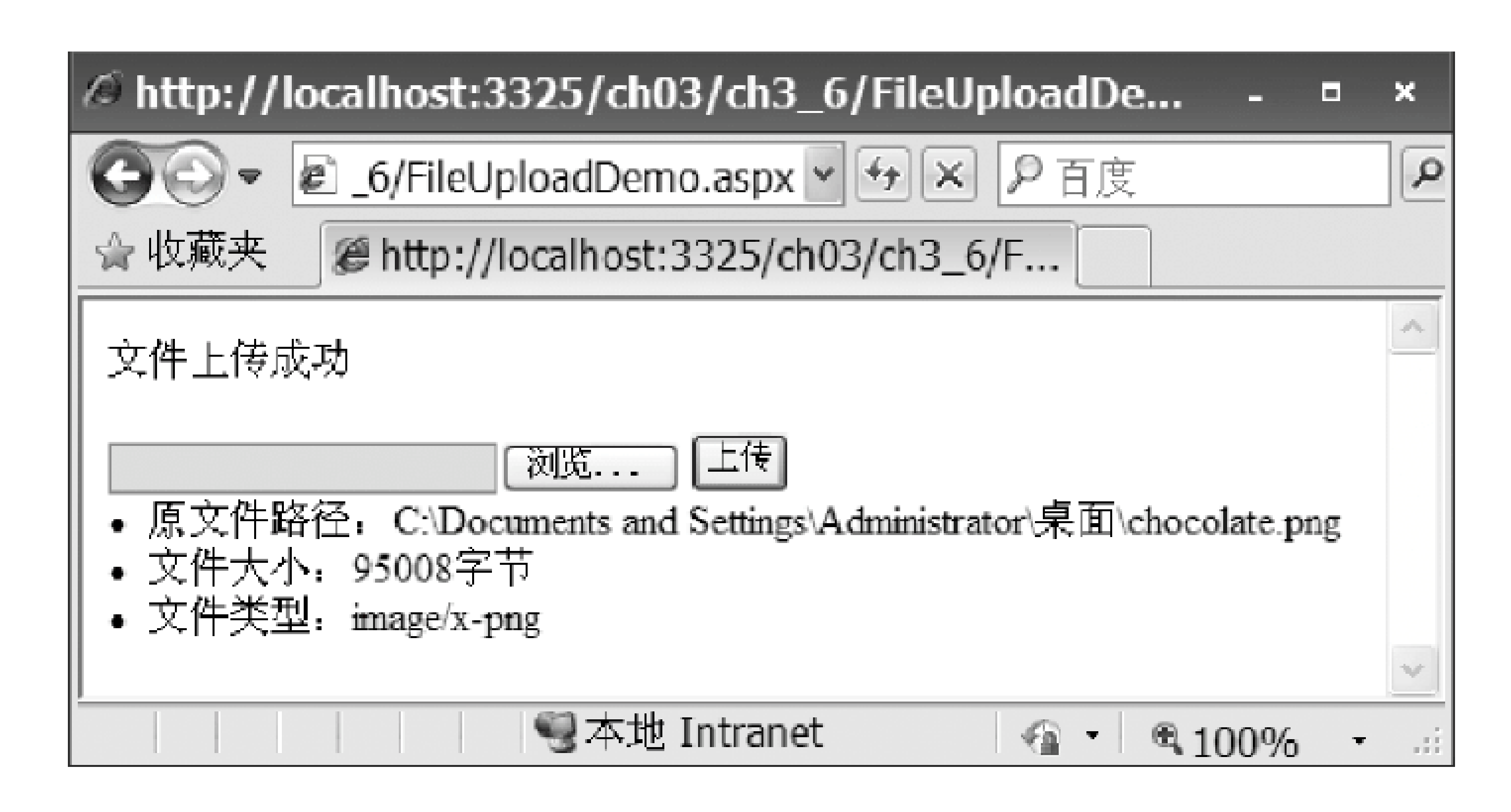


图 3-19　FileUpload 控件运行效果

方控件。

1. 验证码控件 WebValidates

上网的时候，如果登录或者注册，常常需要输入一个验证码，防止竞争对手使用程序模拟在短时间内注册上百万个“脏”用户。验证码通过每次生成不同的验证内容，可以防止基于程序循环而产生的恶意攻击。

要实现验证码，有必要了解验证码的实现方式。首先，验证码是一个图片，包含随机生成的文字。可以使用一个页面，通过程序绘制页面上的内容和干扰像素（又称噪点），然后使用一种状态保持方式，在页面上对比用户输入的内容和刚才生成的内容。这样就可以实现验证码效果。根据不同的状态保持方式，验证方式可以分为 Session 方式和 Cookie 方式。但是这两种方式效果都不是太理想，最麻烦的是还需要编码去绘图。可以使用第三方的验证码控件 WebValidates 来实现验证码效果。

验证码控件的使用步骤如下。

（1）将从网上下载的验证码控件放入工具箱。

（2）拖放控件到页面相应位置。

（3）页面初始化时，编程生成验证码（假设验证码控件 ID 为 snCode）。

（4）编码对比用户的输入（假设用户输入验证码的文本框 ID 是 txtCode），并做相应的处理。

```
snCode.CheckSN(txtCode.Text.Trim())
```

下面使用验证码控件实现用户身份证号码注册功能。

注册页面 Default.aspx 中关键代码如下所示。

```
<form id = "form1" runat = "server">
    <div>
    身份证：<asp:TextBox ID = "txtPhone" runat = "server" Width = "162px"></asp:TextBox><br />
    验证码：<asp:TextBox ID = "txtCode" runat = "server" Width = "159px"></asp:TextBox>
    <cc1:SerialNumber ID = "snCode" runat = "server"></cc1:SerialNumber><br />
    <asp:Button ID = "btnSubmit" runat = "server" OnClick = "btnSubmit_Click" Text = "完 成" />
```

```
        <asp:Label ID="lblMsg" Visible="false" runat="server" Text="Label"></asp:Label>
    </div>
</form>
```

页面的后置代码文件 Default.aspx.cs 中的关键代码如下所示(其中的核心代码以粗体表示)。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        snCode.Create();          //首次加载生成新验证码
    }
}

protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (!CheckCode())             //调用验证方法
    {
        this.lblMsg.Visible = true;
        this.lblMsg .Text = "验证码错误!";
        return;
    }
    else
    {
        this.lblMsg.Visible = true;
        this.lblMsg.Text = "验证码通过!";
    }
}

protected bool CheckCode()        //验证方法
{
    if (snCode.CheckSN(txtCode.Text.Trim()))     //判断验证码输入是否正确
    {
        return true;
    }
    else
    {
        snCode.Create();          //如果验证码输入不正确,则生成新验证码
        return false;
    }
}
```

运行注册页 Default.aspx,效果如图 3-20 所示。

2. 富文本控件 CKeditor

富文本控件就是在线文本编辑控件,可以像 Word 编辑器那样对录入的内容设置样式、排版等,而不用编写 HTML 代码。我们在论坛上发表评论时往往有这样的体验,评论的内



图 3-20　添加验证码后的运行效果

容可以改变字体，可以添加表情图片，可以添加超链接等等。这种功能仅仅使用 TextBox 控件难以实现。常见的在线文本编辑功能的控件有如下几种：

- RichTextBox——最早的富文本控件，富文本控件因它而得名。
- CKeditor——国外一个开源项目。
- CuteEditor——功能最为完善，但它自身也是相当庞大。
- eWebEditor——国产软件，有中国特色。
- FreeTextBox——简单方便，在国内使用相当普遍。

其中，CKeditor 是一款高性能、调用方便以及功能强大的在线编辑器。它支持 ASP.NET、PHP、JSP、Java 等多种语言且不需要用户安装客户端插件。下面以 CKeditor 为例讲解在线编辑录入控件的用法。

1）下载 CKeditor

最新的 CKeditor 可以从 CKeditor 官网 http://www.ckeditor.com 上下载，这里使用最新发布的 CKeditor 3.6.4。下载 ckeditor_aspnet_3.6.4.zip 压缩包并解压后，如图 3-21 所示。打开_Samples 文件夹，有一个包括 CKeditor 所使用的全部图片、JavaScript 脚本等文件的 CKeditor 资源文件；在 bin 目录下的 Debug 文件夹下有一个 CKEditor.NET.dll 文件，提供可以运行在.NET 环境下的程序集。



图 3-21　ckeditor_aspnet_3.6.4 文件结构

2）配置 CKeditor

（1）将 Debug 文件夹下的 CKEditor.NET.dll 文件添加到 Visual Studio 的工具箱中，效果如图 3-22 所示。

（2）将 CKeditor 文件夹复制到网站根目录下。

3）使用 CKeditor

将 CKeditor 拖入页面设计视图中，会自动生成如下所示代码。

```
<%@ Register Assembly = "CKEditor.NET" Namespace = "CKEditor.NET" TagPrefix = "CKEditor" %>
<CKEditor:CKEditorControl ID = "cec" runat = "server" Width = "832px"></CKEditor:CKEditorControl>
```

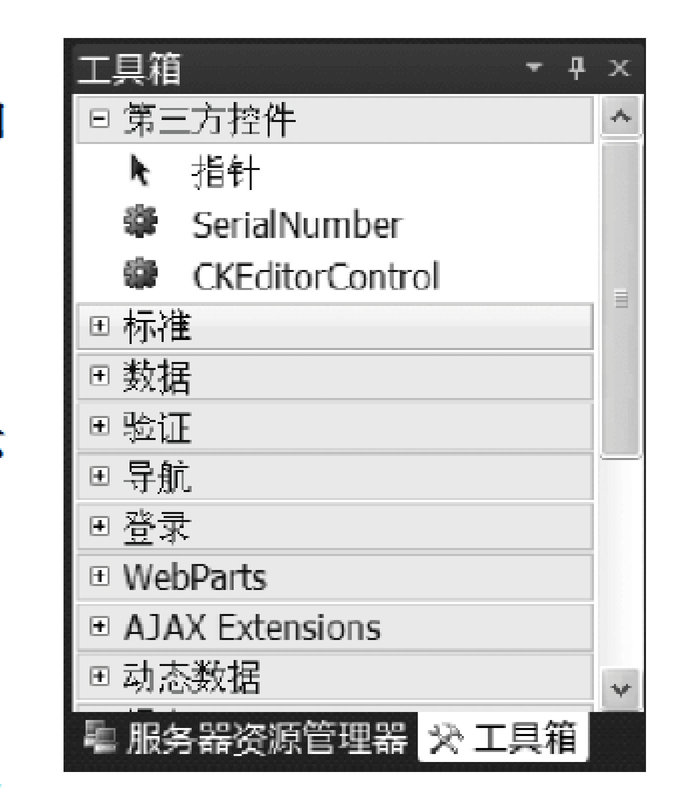


图 3-22　添加 CKEditor 控件到工具栏中的效果

其中，@ Register 指令用于在 ASP.NET 应用程序文件中注册该控件，该指令有几个属性，Assembly 表示使用的程序集，Namespace 表示使用的命名空间，TagPrefix 表示标签的前缀，如“<asp：TextBox>”中的“asp”就是前缀。@ Register 指令下的 CKeditor 的定义要以 CKeditor 作为标签的前缀。

运行页面 CkeditorDemo.aspx，效果如图 3-23 所示。

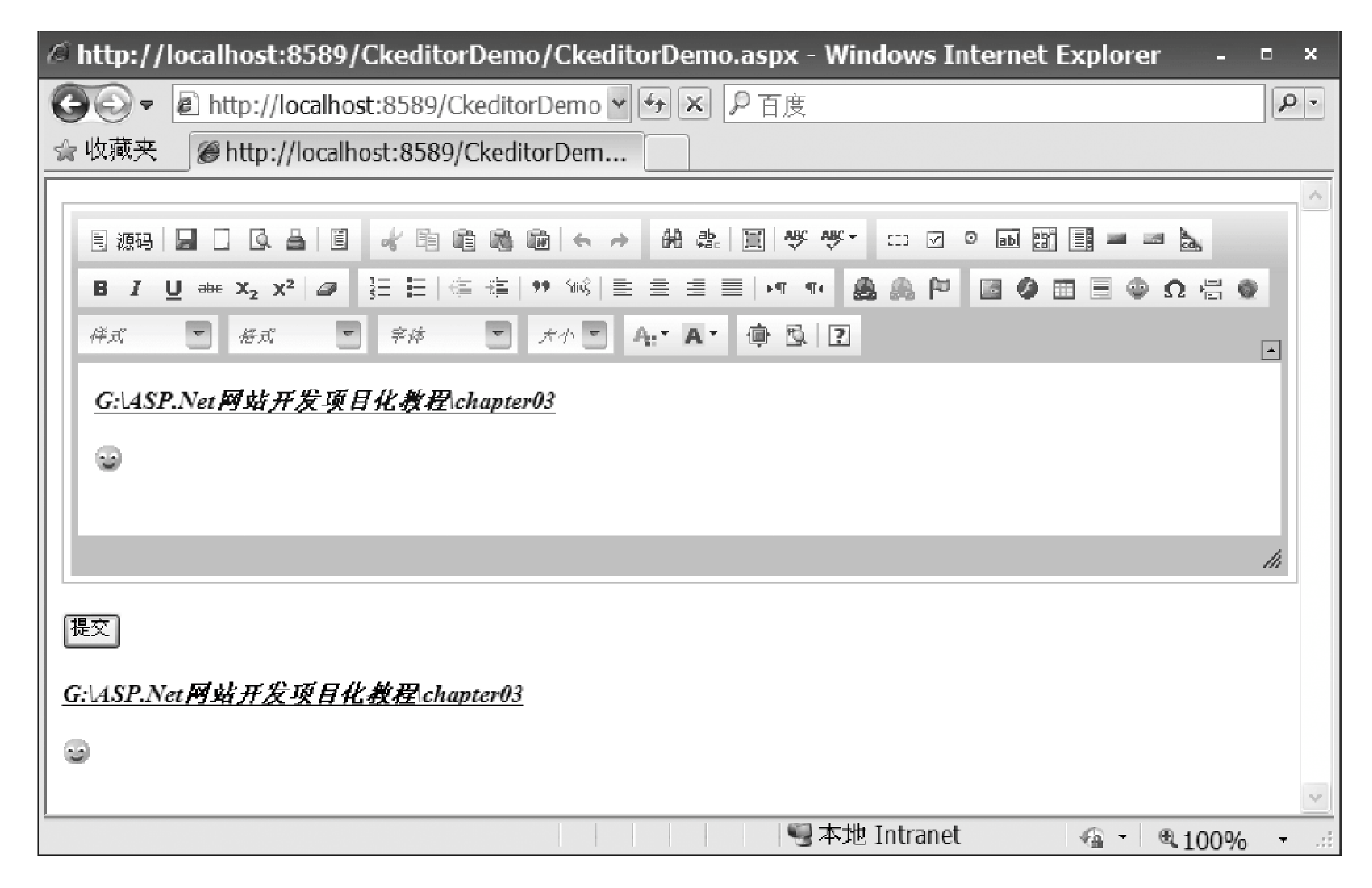


图 3-23　CKeditor 效果图

CKeditor 不同于 TextBox，在 CKeditor 获取输入内容需要使用 Text 属性。

如果在图 3-23 中插入表情、进行样式设置等，那么使用 Text 属性获取到的 CKeditor 控件中的值，内容如下所示。

```
<h1>
    <span style="font-family: times new roman,times,serif">
      <span style="font-size: 16px"><u><em><strong>G:\ASP.Net 网站开发项目化教程
      </strong></em></u>\chapter03</span></span>
</h1>
<p>
    <span style="font-family: times new roman,times,serif"><span style="font-size:
16px"><img alt="smiley"
    height="20" src="http://localhost:8589/CkeditorDemo/ckeditor/plugins/smiley/images/
regular_smile.gif"
    title="smiley" width="20" /></span></span>
</p>
```

其实 CKeditor 存储的是一段 HTML 文本，可以把这段文本直接存入数据库或从数据库读取后显示在页面上，由浏览器本身解析这些图片、表格、字体等。

有时候在编辑过程中会出现如图 3-24 所示的错误。



图 3-24　CKeditor 显示的错误信息

这是由于 ASP.NET 自身的安全机制引起的，它屏蔽了提交有潜在危险的表单。不过该错误信息也有解决办法，在 Page 指令做如下设置就可以了：ValidateRequest="false"。

3. 日期输入控件

在新知书店图书详细页面中，如果手动输入出版日期，易出现格式错误，其用户体验较

差,可以使用日历控件。下面介绍两种常用的日历控件。

1) Calendar 控件

Calendar 控件是 Visual Studio 自带的控件,用于显示一个可选的日历。该控件的常用属性和事件如表 3-14 所示。

表 3-14 Calendar 控件的常用属性和事件

属性和事件		说明
属性	SelectedDate	设置或获取选择的日期
	VisibleDate	当前可见的日期(默认显示月份)
	TitleFormat	标题格式("某月"或"某年某月")
事件	SelectionChanged	选择某日期后的事件

日历控件在用户选择日期后触发 SelectionChanged 事件,可以将用户选择的日期赋给需要的控件。

2) JS 版日历

Calendar 控件有一个缺陷,就是每次日历的显示、隐藏和用户的选择都会造成回传。在 Web 开发中,特别是访问量大的站点开发,往往特别忌讳这些事情。可以选择第三方 JS 版的日历控件。

JS 版的日历控件有多种,它们具有页面无刷新、美观等优点。本书给大家介绍的是 My97DatePicker 日历控件,该控件可以从官网 www.my97.net 免费下载,并且官网中提供了详细的使用说明,这里仅介绍基本的使用方法。

先将下载的文件复制到站点的一个目录中,这里放在根目录下的 My97DatePick 文件夹中,使用时首先在页面中添加如下代码,即引入 JS 文件。

```
<script language = "javascript" type = "text/javascript"
    src = "../My97DatePicker/ WdatePicker.js" >
</script>
```

然后再修改输入如下的日期文本框代码。

```
<form id = "form1" runat = "server">
    <div>
        <asp: TextBox ID = " txtDate" runat = " server" onFocus = " WdatePicker ( )" > </asp:
TextBox>
    </div>
</form>
```

加粗的部分是新添加的代码。当该文本框获得焦点时就显示出日历控件,页面 My97DatePicker.aspx 运行效果如图 3-25 所示。

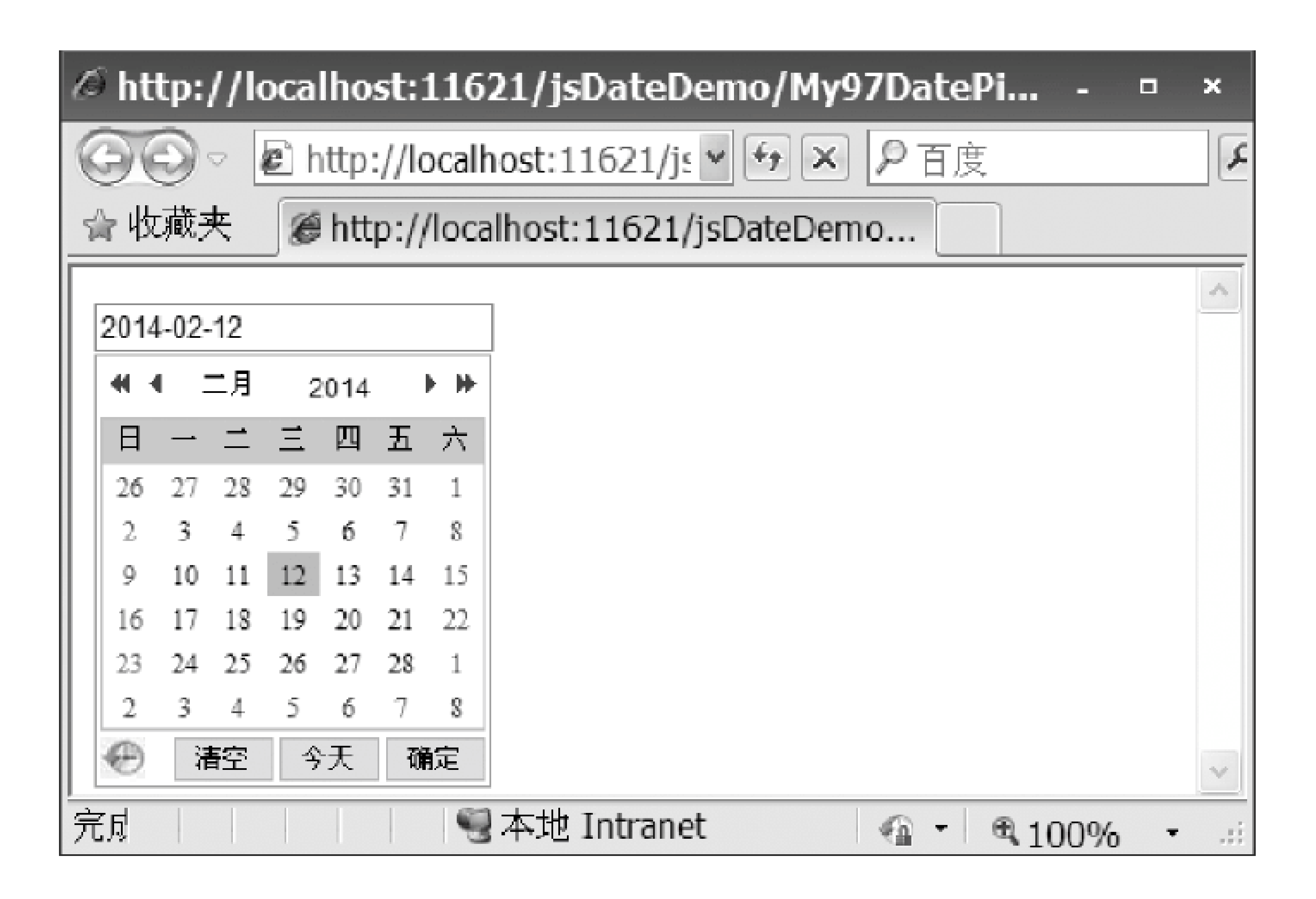


图 3-25 My97DatePicker 的样式

3.2 单元任务

任务 3-2-1 设计"新知书店"用户注册页面

【任务描述】

- 使用控件设计如图 3-26 所示的用户注册页面。
- 为用户注册页面添加验证码功能,验证码输入错误给出提示信息。
- "已经有账号,马上登录"和"这里"是个链接,链接地址设为空。

【任务实施】

(1) 创建网站项目 rw3-2-1,将教学资源包中对应的 Css 及 image 文件夹复制到网站根目录,通过右击网站项目,创建页面 Register. aspx。

(2) 在页面 Register. aspx 的<head runat="server"></head>标签内添加以下代码,用以导入样式文件 member. css 及设置页面标题。

```
<title>新知书店 - 最方便的网上书店</title>
<link href = "Css/member.css" rel = "stylesheet" type = "text/css" />
```

(3) 切换至 Register. aspx 页面设计视图,如图 3-26 所示,从工具箱拖入 Label 标签、TextBox、Button 控件和第三方控件——验证码控件 SerialNumber 至页面相应位置,并切换至源视图,调整编写代码如下。

```
<form id = "form1" runat = "server">
  <div id = "action_area" class = "member_form">
    <h2 class = "action_type"><img src = "Images/register.gif" alt = "会员注册" /></h2>
    <p>
```

```
            <label><span>*</span>用户名</label>
            <asp:TextBox CssClass="opt_input" ID="txtLoginId" runat="server"></asp:TextBox>
        </p>
        <p>
            <label><span>*</span>真实姓名</label>
            <asp:TextBox CssClass="opt_input" ID="txtName" runat="server"></asp:TextBox>
        </p>
        <p>
            <label><span>*</span>密     码</label>
            <asp:TextBox CssClass="opt_input" ID="txtLoginPwd" runat="server"
                  TextMode="Password"></asp:TextBox></p>
        <p>
            <label><span>*</span>确认密码</label>
            <asp:TextBox CssClass="opt_input" ID="txtPwdAgain" runat="server"
                  TextMode="Password"></asp:TextBox>
        </p>
        <p>
            <label><span>*</span>Email</label>
            <asp:TextBox CssClass="opt_input" ID="txtEmail" runat="server"></asp:TextBox>
        </p>
        <p>
            <label><span>*</span>地址</label>
            <asp:TextBox CssClass="opt_input" ID="txtAddress" runat="server"></asp:TextBox>
        </p>
        <p>
            <label><span>*</span>手机</label>
            <asp:TextBox CssClass="opt_input" ID="txtTele" runat="server"></asp:TextBox>
        </p>
        <p>
            <label><span>*</span>验证码</label>
            <asp:TextBox CssClass="opt_input" ID="txtCode" runat="server"></asp:TextBox> 
            <cc1:SerialNumber ID="snCode" runat="server"></cc1:SerialNumber>
            <asp:Label ID="lblMsg" runat="server" Text="Label" Visible="false"></asp:Label>
        </p>
        <p class="form_sub">
            <asp:Button ID="btnRegister" runat="server" Text="确定了,马上提交"
                  CssClass="opt_sub"></asp:Button></p>
        <p class="form_sub"> 加<span>*</span>的为必填项目</p>
        <p class="form_sub">
            <a href="#">已经有账号,马上登录</a><br />
            如果你已经有"新知书店"社区账号,请点<a href="javascript:alert('书店社区暂未开通');">
            这里</a>登录升级
          </p>
      </div>
    </form>
```

（4）编写页面后置文件 Register.aspx.cs 中实现验证码的生成代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
```

```
            snCode.Create();                        //首次加载时生成验证码
        }
    }

    protected bool CeckCode()
    {
        if (snCode.CheckSN(txtCode.Text.Trim()))    //判断验证码输入是否正确
        {
            return true;
        }
        else
        {
            snCode.Create();                        //如果验证码输入不正确,则生成新的验证码
            return false;
        }
    }
```

（5）运行页面 Register.aspx,效果如图 3-26 所示。

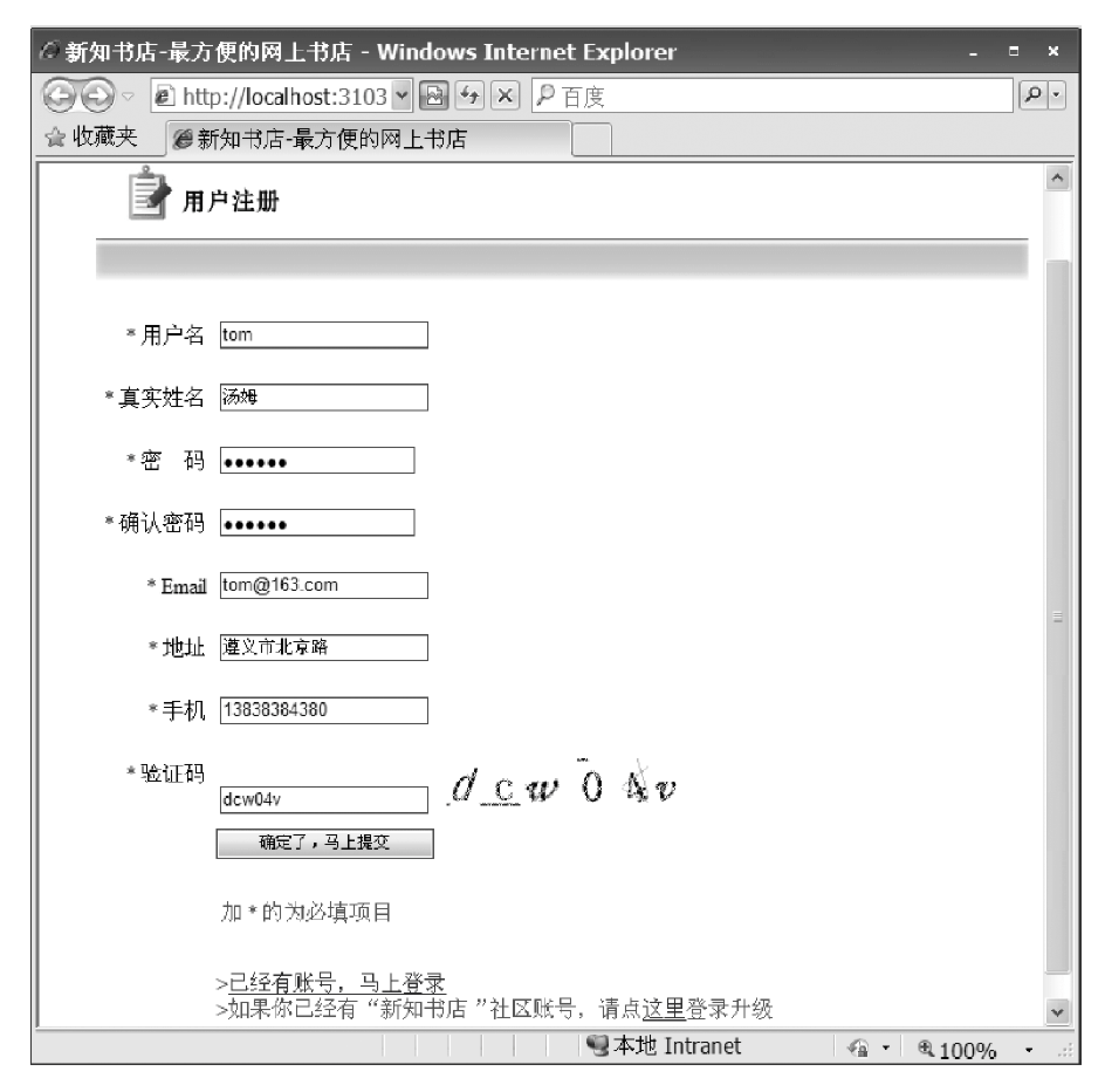


图 3-26　“新知书店”用户注册页面

任务 3-2-2 为“新知书店”注册页面添加回车自动提交功能

【任务描述】

在任务 3-2-1 基础上添加回传自动提交功能，具体要求如下：

- 当用户输入验证码时，判断用户按下的键，如果用户按下的是 Enter 键，则给出“确定全部提交吗?”的提示。
- 出现提示后，如果用户单击“确定”按钮，则提交表单，效果如图 3-27 所示。
- 使用 literal 控件给出验证码输入结果的提示信息。



图 3-27 新知书店用户注册页的回车自动提交功能

【任务实施】

(1) 创建网站项目 rw3-2-2。由于是在上一个任务的基础上来完成，故将任务 3-2-1 目录下的全部内容复制到网站项目 rw3-2-2 下。

(2) 当页面加载时向验证码输入框的 Attributes 属性添加 onkeydown 事件，代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    //其他代码省略
```

```
        this.txtCode.Attributes.Add("onkeydown", " return SubmitClick()");
    }
```

上述代码的本质是在运行时给控件 txtCode 添加客户端事件，其中 SubmitClick()为页面 Register.aspx 中<Script>块中的事件方法，代码如下。

```
    < script type = "text/javascript" language = "javascript">
        function SubmitClick() {
            if (event.keyCode == 13) {
                if (!confirm("确定全部提交吗?")) {
                    return false;
                }
                event.keyCode = 9;
                event.returnValue = false;
                document.getElementById("btnRegister").click();    //响应"提交"按钮单击事件
                return true;
            }
        }
    </script >
```

这段代码使用 event.keyCode 来判断用户的键盘操作(13 即表示 Enter 键)。

(3) 使用 literal 控件给出验证码输入结果的提示信息，代码如下。

```
    protected void btnRegister_Click(object sender, EventArgs e)
    {
        if (!CheckCode())              //调用验证方法
        {
            /* Page.ClientScript .RegisterClientScriptBlock(this.GetType (),"alert",
                                      "< script > alert('验证码错误!')</script >"); */
            this.ltScript.Text = "验证码错误!";
            return;
        }
        else
        {
            /* Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "alert",
                                      "< script > alert('验证码正确!')</script >"); */
              this.ltScript.Text = "< sript > Alert('验证码正确!')</sript >";
        }
    }
```

Literal 控件是一个简化的 Label 控件，可以显示文本或 HTML 内容，使用 Literal 控件可以实现以编程方式设置文本而不需要添加任何额外的 HTML 标记，使用非常简单。

注意：*使用 Response.write()方法也可以实现以编程方式设置文本，但由于 Response.write()方法是在执行 HTML 页面代码之前就输出执行的，所以输出的脚本在页面的顶部显示，从而破坏了页面布局。*

除了可以使用 Label 控件在相应位置显示提示信息外，还可以用以下代码显示“弹出式”提示信息。

```
    Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "alert", "< script > alert('验证
    码正确!')</script >");
```

我们不去具体分析这段代码,大家可以在需要的时候调用即可。

(4) 运行页面 Register.aspx,输入验证码,按 Enter 键,并且在图 3-27 提示对话框中单击"确定"按钮,则执行"确定了,马上提交"按钮单击事件。

任务 3-2-3　为"新知书店"用户注册页面添加验证功能

【任务描述】

在任务 3-2-2 的基础上实现"新知书店"用户注册页的验证功能,效果如图 3-28 所示,具体符合以下要求。

- 所有输入文本框均要有非空验证,如果为空给出"请输入 *** "的提示。
- "密码"和"确认密码"要求输入一致,不一致给出"两次密码不一致"的提示。
- "邮件"和"手机"的格式要求输入正确,其中"手机"要求输入位数为 11 位。
- 所有的错误信息以弹出信息框的方式汇总显示。

【任务实施】

(1) 运行 Visual Studio 2010,新建网站项目 rw3-2-3,将任务 3-2-2 中的所有文件复制到新建的网站项目中。

(2) 从工具箱中拖放非空验证控件 RequiredFieldValidator、CompareValidator、RegularExpressionValidator、ValidationSummary 至页面 Register.aspx 相应文本框控件对应的位置,添加代码如下。

```
<form id="form1" runat="server">
  <div id="action_area" class="member_form">
    <h2 class="action_type"><img src="Images/register.gif" alt="会员注册" /></h2>
    <p>
        <label><span>*</span>用户名</label>
        <asp:TextBox CssClass="opt_input" ID="txtLoginId" runat="server"></asp:
TextBox>
        <asp:RequiredFieldValidator ID="valrLoginId" runat="server" ErrorMessage="请输
入用户名"
              ControlToValidate="txtLoginId">*</asp:RequiredFieldValidator>
    </p>
    <p>
        <label><span>*</span>真实姓名</label>
        <asp:TextBox CssClass="opt_input" ID="txtName" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="valrName" runat="server" ErrorMessage="请输入真
实姓名"
              ControlToValidate="txtName">*</asp:RequiredFieldValidator>5-12 个字符或数字
    </p>
    <p>
        <label><span>*</span>密     码</label>
        <asp:TextBox CssClass="opt_input" ID="txtLoginPwd" runat="server"
              TextMode="Password"></asp:TextBox>
        <asp:RequiredFieldValidator ID="valrPass" runat="server" ErrorMessage="请输入密码"
              ControlToValidate="txtLoginPwd">*</asp:RequiredFieldValidator></p>
    <p>
```

```
        <label><span>*</span>确认密码</label>
        <asp:TextBox CssClass="opt_input" ID="txtPwdAgain" runat="server"
            TextMode="Password"></asp:TextBox>
        <asp:CompareValidator ID="valcPwd" runat="server" ErrorMessage="两次密码不一致"
            ControlToCompare="txtLoginPwd"
            ControlToValidate="txtPwdAgain">*</asp:CompareValidator>
    </p>
    <p>
        <label><span>*</span>Email</label>
        <asp:TextBox CssClass="opt_input" ID="txtEmail" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="valrEmail" runat="server" ErrorMessage="请输入
Email"
            ControlToValidate="txtEmail">*</asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator ID="valeEmail" runat="server"
            ErrorMessage="Email格式错误" ControlToValidate="txtEmail"
            ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]
\w+)*">*
            </asp:RegularExpressionValidator>
    </p>
    <p>
        <label><span>*</span>地址</label>
        <asp:TextBox CssClass="opt_input" ID="txtAddress" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="valrAddr" runat="server" ErrorMessage="请输入地址"
            ControlToValidate="txtAddress">*</asp:RequiredFieldValidator>
    </p>
    <p>
        <label><span>*</span>手机</label>
        <asp:TextBox CssClass="opt_input" ID="txtTele" runat="server"></asp:TextBox>
        <asp:RequiredFieldValidator ID="valrTel" runat="server" ErrorMessage="请输入手机号"
            ControlToValidate="txtTele">*</asp:RequiredFieldValidator>
        <asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
            ValidationExpression="\d{11}" ControlToValidate="txtTele"
            ErrorMessage="手机号位数不正确">*</asp:RegularExpressionValidator>
    </p>
    <p>
        <label><span>*</span>验证码</label>
        <asp:TextBox CssClass="opt_input" ID="txtCode" runat="server"></asp:TextBox> 
        <cc1:SerialNumber ID="snCode" runat="server"></cc1:SerialNumber>
    </p>
    <asp:ValidationSummary ID="valsRegister" runat="server" ShowMessageBox="True"
        ShowSummary="False" />
    <p class="form_sub">
        <asp:Button ID="btnRegister" OnClick="btnSubmit_Click" runat="server" Text="确
定了,马上提交"
          CssClass="opt_sub"></asp:Button></p>
    <p class="form_sub"> 加<span>*</span>的为必填项目</p>
    <p class="form_sub">
        <a href="">已经有账号,马上登录</a><br />
        如果你已经有"新知书店"社区账号,请点<a href="javascript:alert('书店社区暂未开通');">
```

```
            这里</a>登录升级</p>
      </div>
   </form>
```

使用正则表达式控件 RegularExpressionValidator 完成“邮件”和“手机”的验证，手机位数验证的正则表达式为“d{11}”，ValidationSummary 用于以弹出窗口方式汇总所有错误报告。

提示：有时候你会发现 ValidationSummary 显示错误时，在验证控件的位置还是显示了错误提示信息。这时用户可以设置验证控件的 Text 属性为“*”，那样就会在错误信息提示的时候，在验证控件的位置仅显示一个红色的“*”；还有一种方式，就是不设置 Text 属性，而是在验证控件的标签中写“*”，效果是一样的。

(3) 运行页面 Register.aspx，在文本框中输入相应信息，当手机号码只输了 10 位，效果如图 3-28 所示。

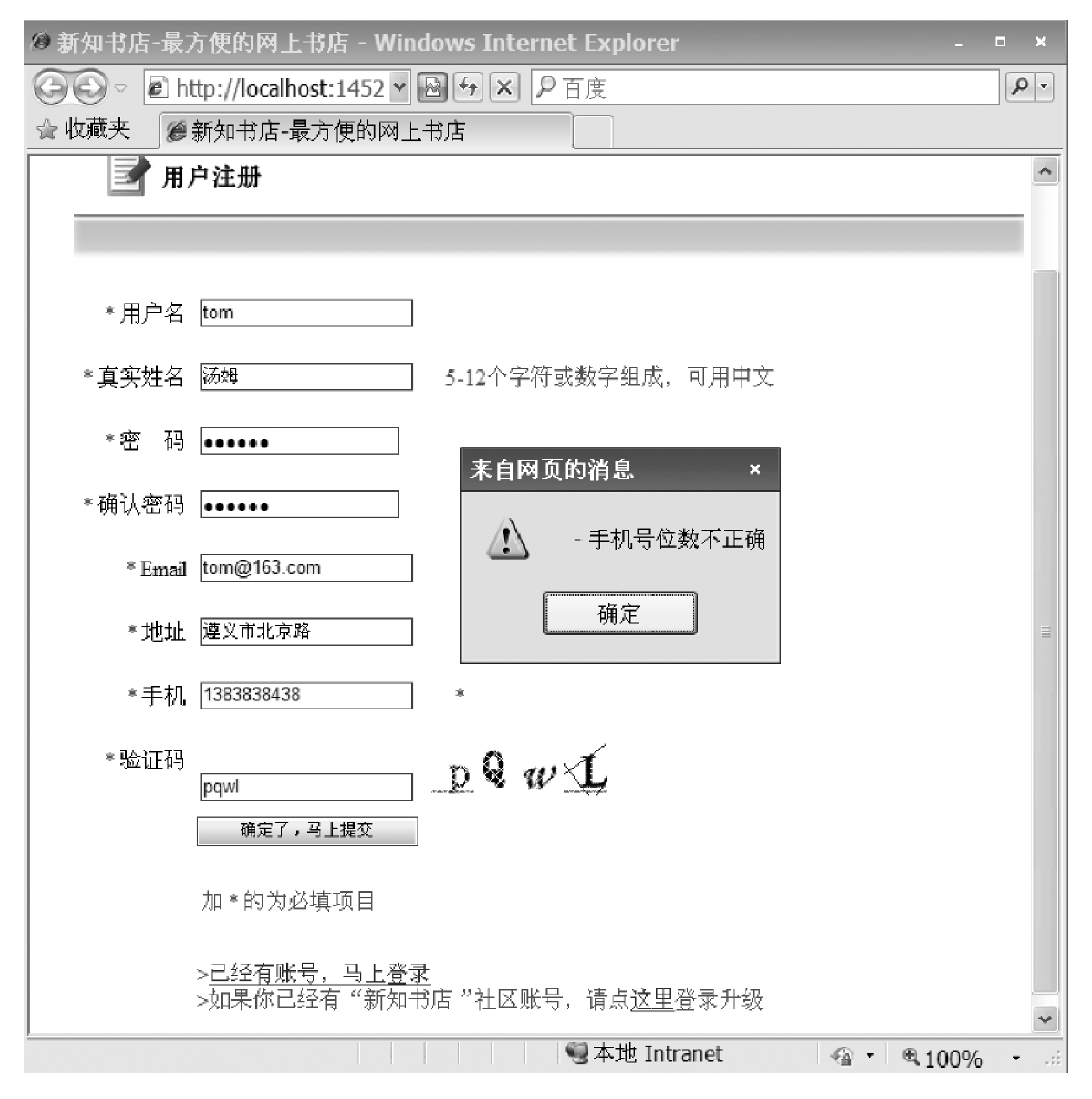


图 3-28 新知书店用户注册页面验证功能

若手机号码也按要求输入正确的11位数字，则不会有如图3-28所示的弹出式错误提示。

3.3 项目实训

设计“博客系统”的会员注册页面

【需求说明】

- 使用控件设计如图3-29所示的“博客系统”会员注册页面。
- 为会员注册页面添加验证码功能，验证码输入错误给出提示信息。

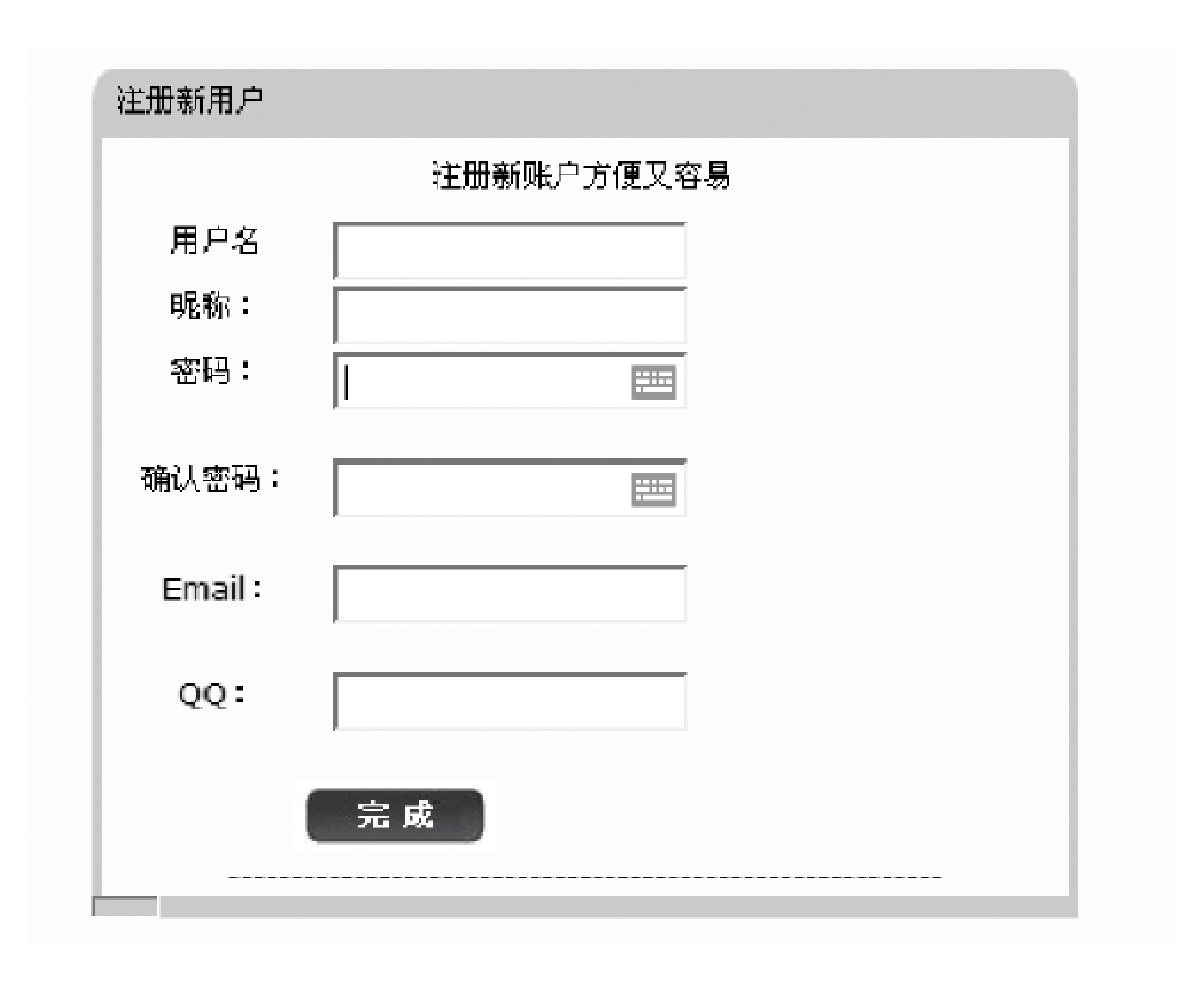


图3-29 “博客系统”会员注册页面效果图

3.4 单元小结

本单元主要介绍了运行在服务器端的HTML服务器控件，ASP.NET控件的使用(控件的主要属性、事件驱动机制等)，使用其设计出较好满足交互性要求的Web页面，使读者理解Web窗体页面服务器控件的功能以及与传统HTML控件的区别。还重点介绍了服务器控件——数据验证控件，能够方便地完成页面输入数据的验证功能。此外，还介绍了功能强大的文件上传控件及非常实用的验证码、富文本控件等第三方控件。ASP.NET控件中的控件知识体系如图3-30所示(▦表示没有详细介绍，读者可以查阅相关资料了解更多信息，下同)。

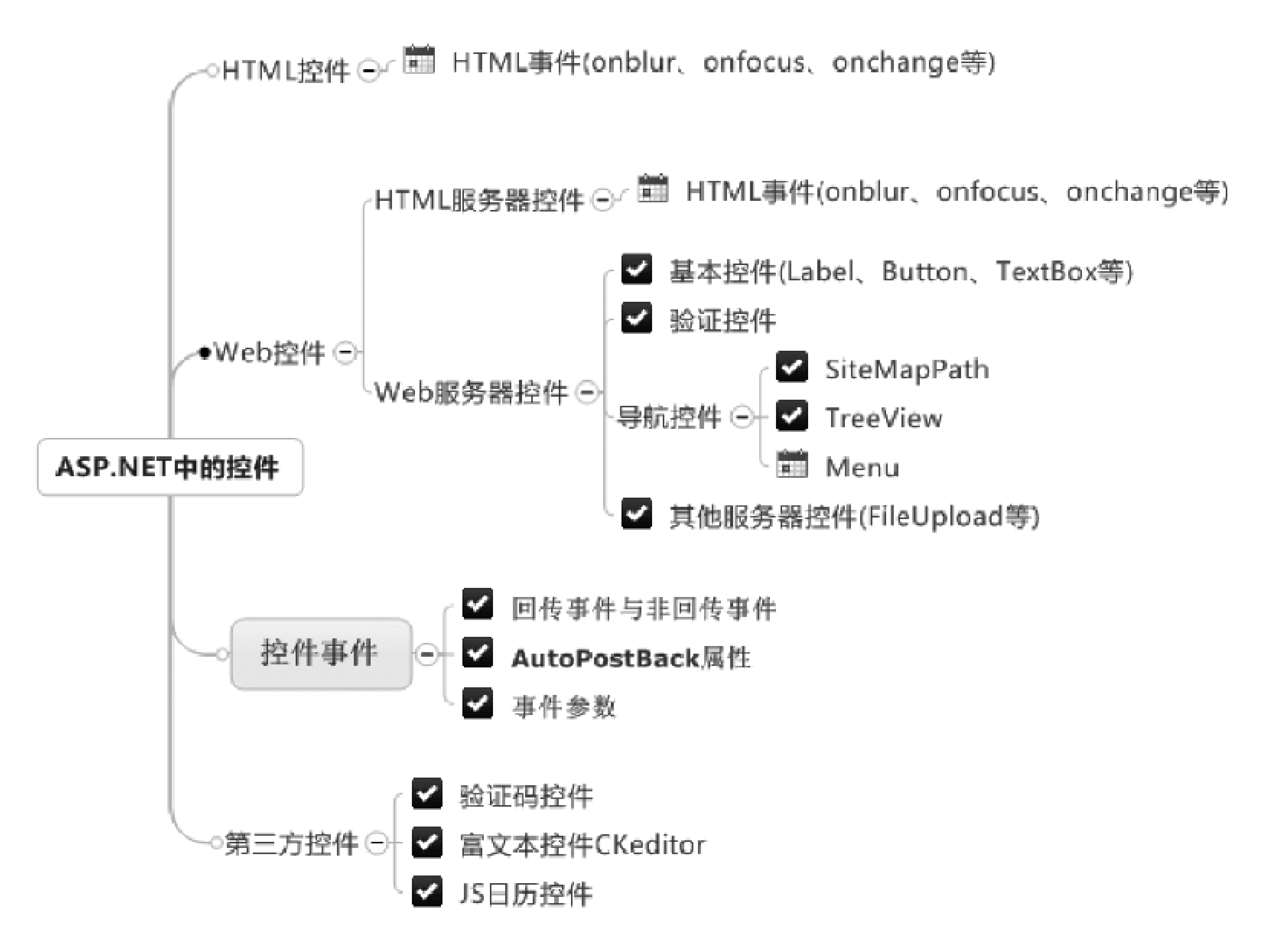


图 3-30 ASP.NET 中的控件知识体系

3.5 单元练习题

一、选择题

1. 在 Web 窗体中,放置一个 HTML 控件,采用下列(　　)方法变为 HTML 服务器控件。

A. 添加 runat="server"和设置 Attribute 属性

B. 添加 id 属性和 Attribute 属性

C. 添加 runat="server"和设置 id 属性

D. 添加 runat="server"和设置 Value 属性

2. 要把一个 TextBox 设置成密码输入框,应该设置(　　)属性。

A. Columns　　B. Rows　　C. Text　　D. TextMode

3. 下面(　　)控件不包含 ImageUrl 属性。

A. HyperLink　　B. Image　　C. ImageButton　　D. LinkButton

4. AlternateText 属性是(　　)控件特有的属性。

A. HyperLink　　B. Image　　C. ListBox　　D. LinkButton

5. 添加一个服务器 CheckBox 控件,单击该控件不能生成一个回传,如何做才能让 CheckBox 的事件导致页面被提交?(　　)

A. 设置 IE 浏览器可以运行脚本　　B. AutoPostBack 属性设置为 true

C. AutoPostBack 属性设置为 false　　D. 为 CheckBox 添加 Click 事件

6. 如果希望控件的内容变化后,立即回传页面,需要在控件中添加(　　)属性。

A. AutoPostBack="true"　　B. AutoPostBack="false"

C. IsPostBack="true"　　D. IsPostBack ="false"

7. 在下面的控件中,(　　)可以将其他控件包含在其中,所以它常常用来包含一组控件。

A. Calendar　　B. Button　　C. Panel　　D. DropDownList

8. 下面对服务器验证控件说法正确的是(　　)。

A. 可以在客户端直接验证用户输入,并显示出错消息

B. 服务器验证控件种类丰富,共有十种之多

C. 服务器验证控件只能在服务器端使用

D. 各种验证控件不具有共性,各自完成功能

9. 用户登录界面中要求用户必须填写用户名和密码,才能提交,应使用(　　)控件。

A. RequiredFieldValidator　　B. RangeValidator

C. CustomValidator　　D. CompareValidator

10. 在一个注册界面中,包含用户名、密码、身份证三项注册信息,并为每个控件设置了必须输入的验证控件。但为了测试的需要,要暂时取消该页面的验证功能,该如何做?(　　)

A. 将提交按钮的 CausesValidation 属性设置为 true

B. 将提交按钮的 CausesValidation 属性设置为 false

C. 将相关的验证控件属性 ControlToValidate 设置为 true

D. 将相关的验证控件属性 ControlToValidate 设置为 false

11. 现有一课程成绩输入框,成绩范围为 0～100,这里最好使用(　　)验证控件。

A. RequiredFieldValidator　　B. CompareValidator

C. RangeValidator　　D. RegularExpressionValidator

12. 如果需要确保用户输入大于 30 的值,应该使用(　　)验证控件。

A. RequiredFieldValidator　　B. CompareValidator

C. RangeValidator　　D. RegularExpressionValidator

13. RegularExpressionValidator 控件中可以加入正则表达式,下面选项对正则表达式说法正确的是(　　)。

A. "."表示任意数字

B. "*"表示和其他表达式一起,表示任意组合

C. "\d"表示任意字符

D. "[A-Z]"表示 A～Z 顺序的大写字母

14. 下面对 CustomValidator 控件说法错误的是(　　)。

A. 控件允许用户根据程序设计需要自定义控件的验证方法

B. 控件可以添加客户端验证方法和服务器端验证方法

C. ClientValidationFunction 属性指定客户端验证方法

D. runat 属性用来指定服务器端验证方法

15. 使用 ValidationSummary 控件时需要以对话框的形式来显示错误信息,需要设置下面(　　)属性。

A. 设置 ShowSummary 为 true　　B. 设置 ShowMessage 为 true

C. 设置 ShowMessage 为 false　　D. 设置 ShowSummary 为 false

16. 创建一个 Web 窗体,其中包括多个控件,并添加了验证控件进行输入验证,同时禁

止所有客户端验证。当单击按钮提交窗体时，为了确保只有当用户输入的数据完全通过验证时才执行代码处理，需如何处理？(　　)

A. 在 Button 控件的 Click 事件处理程序中，测试 Page.IsValid 属性，如果该属性为 true 则执行代码

B. 在页面的 Page_Load 事件处理程序中，测试 Page.IsValid 属性，如果该属性为 true 则执行代码

C. 在 Page_Load 事件处理程序中调用 Page 的 Validate 方法

D. 为所有的验证控件添加 runat="server"

二、填空题

1. RadioButtonList 服务器控件的________属性决定单选按钮是水平还是垂直方式显示，________属性可以获取或设置在 RadioButtonList 控件中显示的列数。

2. 使用________控件可以在页面上显示一个日历。

3. 如果希望将特定的输入控件与另一个输入控件相比较，需要使用________验证控件。

4. RangeValidator 控件中，通过________属性指定要验证的输入控件；MinimumValue 属性指定有效范围的最小值；________属性指定有效范围的最大值；Type 属性用于指定要比较的值的数据类型。

5. 验证 6 位数字的正则表达式是________。

6. 通过________控件验证用户是否在文本框中输入了数据；通过 CompareValidator 控件将输入控件的值与常数值或其他输入控件的值相比较，以确定这两个值是否与比较运算符(小于、等于、大于)指定的关系相匹配；通过________控件可以自定义验证规则；________控件用于罗列网页上所有验证控件的错误消息。

三、问答题

1. Button、LinkButton 和 ImageButton 控件有什么共同点？

2. 比较 ListBox 和 DropDownList 控件的相同点和不同点。

3. 验证控件有几种类型？分别写出它们的名称。

4. 验证控件的 ErrorMessage 和 Text 都可以设置验证失败时显示的错误信息，两者有什么不同？

5. 在使用 RangeValidator 控件或 CompareValidator 控件时，如果相应的输入框中没有输入内容，验证是否能够通过？

6. 什么是第三方控件？列举出三个常用的第三方控件并分别简述它们的作用。

单元4 系统对象与数据传递

教学目标：

- 会获取客户端数据与跨页传递数据。
- 掌握 Request、Response、Session、Application、Server 和 Cookie 等对象的作用、常用属性和方法。
- 掌握多种页面跳转的实现方法与数据编码。
- 会应用 Application 对象、Session 对象和全局应用程序类。
- 会应用 Application 对象、Session 对象和 Cookie 对象。
- 熟悉 Page 对象和@Page 指令，了解 Page 对象的生命周期和常规 Web 页面生命周期阶段。

4.1 知识准备

4.1.1 ASP.NET 对象概述及属性方法事件

所谓对象(Object)，可以泛指日常生活中看到的和看不到的一切事物，在程序设计中可以用一种仿真的方式来表示对象，一般的对象都有一些静态的特征，如对象的外观、大小等，这在面向对象程序(OOP)中就是对象的属性(attribute)，一般的对象如果是有生命、可以动作的，在面向对象程序中就是对象的方法(method)，所以在面向对象程序的概念中，对象有两个重点：一个是“属性”，另一个是“方法”。

一般而言，对象的定义就是每个对象都具有不同的功能与特征，不同对象属于不同的类(Class)，类定义了对象的属性、方法和事件等特征，没有类就没有对象。

- 属性代表对象的状态、数据和设置值。属性的设置语法如下：

```
对象名.属性名 = 语句(一般又叫属性值)
```

- 方法可以执行动作。方法的调用语法如下：

```
对象名.方法(参数)
```

- 事件的概念的概念比较抽象，通常是一个执行的动作，也就是对象所认识的动作，事件的执行由对象触发。

ASP.NET 中的 Page、Response、Request 等对象(见表 4-1),由.NET Framework 中封装好的类来实现,并且由于这些对象在 ASP.NET 页面初始化请求时自动创建,所以能在程序中的任何地方直接调用,而无须对类进行实例化操作。

表 4-1 ASP.NET 常用内部对象简要说明

对象	功　能
Page	页面对象,用于整个页面的操作
Request	提供对当前页请求的访问,其中包括请求标题、Cookie、客户端证书、查询字符串等,可以用它来读取浏览器已经发送的内容
Response	提供对输出流的控制,如可以向浏览器输出信息、Cookie 等
Session	为当前用户会话提供信息。还提供对可用于存储信息的会话范围的缓存的访问以及控制如何管理会话的方法
Application	提供对所有会话的应用程序范围的方法和事件的访问,还提供对可用于存储信息的应用程序范围的缓存的访问
Server	提供用于在页之间传输控件的实用方法,获取有关最新错误的信息,对 HTML 文本进行编码和解码,获取服务器信息等
Cookie	用于保存 Cookie 信息

下面将分别介绍这些对象的常用属性及方法。

4.1.2 Page 对象

1. 页面对象(Page 对象)生命周期

我们项目中的所有 Web 页面都继承于 System. Web. UI. Page 类,Web 页面运行时,将经历一个生命周期,在生命周期中将执行一系列处理步骤。这些步骤包括初始化、实例化控件、还原和维护状态、运行事件处理程序代码以及进行呈现。了解页生命周期非常重要,因为这样做就能在生命周期的合适阶段编写代码,以达到预期效果。页面生命周期阶段如表 4-2 所示。

表 4-2 常规页生命周期阶段

阶段	说　明
页请求	页请求发生在页生命周期开始之前。用户请求页时,ASP.NET 将确定是否需要分析和编译页(从而开始页的生命周期),或者是否可以在不运行页的情况下发送页的缓存版本以进行响应
开始	在开始阶段,将设置页属性,如 Request 和 Response。在此阶段,页还将确定请求是回传请求还是新请求,并设置 IsPostBack 属性。此外,在开始阶段,还将设置页的 UICulture 属性
页初始化	页初始化期间,可以使用页中的控件,并将设置每个控件的 UniqueID 属性。此外,任何主题都将应用于页。如果当前请求是回传请求,则回传数据尚未加载,并且控件属性值尚未还原为视图状态中的值
加载	加载期间,如果当前请求是回传请求,则将使用从视图状态和控件状态恢复的信息加载控件属性

续表

阶段	说　明
回传事件处理	如果请求是回传请求，则将调用控件事件处理程序。之后，将调用所有验证程序控件的 Validate 方法，此方法将设置各个验证程序控件和页的 IsValid 属性
呈现	在呈现之前，会针对该页和所有控件保存视图状态。在呈现阶段中，页会针对每个控件调用 Render 方法，它会提供一个文本编写器，用于将控件的输出写入页的 Response 属性的 OutputStream 中
卸载	完全呈现页并已将页发送至客户端，准备丢弃该页时，将调用卸载。此时，将卸载页属性（如 Response 和 Request）并执行清理

2. Page 类的主要属性、方法和事件

Page 类与扩展名为.aspx 的文件相关联，这些文件在运行时被编译为 Page 对象，并被缓存在服务器内存中。Page 类的常见属性和方法如表 4-3 所示。

表 4-3　Page 类的重要属性和方法

属性和方法	说　明
Application	为当前 Web 请求获取 HttpApplicationState 对象
Buffer	基础结构。设置指示是否对页输出进行缓冲处理的值
IsPostBack	获取一个值，该值指示页面是首次加载和访问，还是为了响应客户端回传而加载
IsValid	获取一个值，该值指示页验证是否成功
Request	获取请求的页的 HttpRequest 对象
Response	获取与该 Page 对象关联的 HttpResponse 对象。该对象使你得以将 HTTP 响应数据发送到客户端，并包含有关该响应的信息
Server	获取 Server 对象，它是 HttpServerUtility 类的实例
Session	获取 ASP.NET 提供的当前 Session 对象
Validators	获取请求的页上包含的全部验证控件的集合
ViewState	获取状态信息的字典，这些信息使你可以在同一页的多个请求间保存和还原服务器控件的视图状态
DataBind()	将数据源绑定到被调用的服务器控件及其所有子控件
Dispose()	使服务器控件得以在从内存中释放之前执行最后的清理操作

Page 类中很多属性是对象的引用，比如表 4-3 中的 Request、Response、Application 和 Session 等属性，这样在页面中可以直接对这些对象进行访问，而无须通过 Page 对象。比如下面两行代码的作用是一样的。

```
Page.Response.Redirect("Default.aspx");
Response.Redirect("Default.aspx");
```

上述代码第 1 行通过 Page 对象的 Response 属性得到 Response 对象的引用，第 2 行直接通过 Response 对象名对 Response 对象进行引用。

Page 类除了属性和方法之外，还有 8 个常见的事件，如表 4-4 所示。

表 4-4 Page 类的主要事件

事件名称	说明
PreInit	在页初始化开始时发生，是页面执行时第一个被触发的事件
PreLoad	在页面 Load 事件之前发生，即在信息被写入到客户端前触发此事件
Load	当服务器控件(网页)加载到 Page 对象中时触发
Init	当服务器控件(网页)初始化时发生；初始化是控件生存期的第一步
PreRender	在信息被写入到客户端前会触发此事件
Unload	当网页从内存中卸载时，即信息被写入到客户端后触发此事件
InitComplete	在页初始化完成时发生
LoadComplete	在页生命周期的加载阶段结束时发生

表 4-4 中 Page 对象的事件贯彻于页面执行的整个过程。在每个阶段，ASP.NET 都触发了可以在代码中处理的事件，对于大多数情况，只需关心 Page_Load 事件，即：Page_Load(object sender, EventArgs e)。该事件的两个参数是由 ASP.NET 定义的，第 1 个参数定义了产生事件的对象，第 2 个参数是传递给事件的详细信息。每次触发服务器控件时，页面都会去执行一次 Page_Load 事件，说明页面被加载了一次，这个技术称为回传(或者称为回送)技术，是 ASP.NET 最为重要的特征之一。

在 ASP.NET 中，当客户端触发了一个事件，它不是在客户端浏览器上对事件进行处理，而是把该事件的信息传送回服务器进行处理。服务器在接收到这些信息后，会重新加载 Page 对象，然后处理该事件，所以 Page_Load 事件被再次触发。

IsPostBack 属性表示页面是否被首次加载和访问。当 IsPostBack 为 true，表示该请求是为响应客户端回传而加载；当 IsPostBack 为 false，表示该页是被首次加载和访问。如：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        Response.Write("页面首次加载!");
    }
    else
    {
      Response.Write("页面响应客户端回传而加载!");
    }
}
```

注意：由于 Page_Load 事件在每次页面加载时运行，因此其中的代码即使在回传的情况下也会被运行，这时 Page 的 IsPostBack 属性就可以用来解决这个问题，因为这个属性是用来识别 Page 对象是否处于一个回传的状态下，也就弄清楚是请求页面的一个实例，还是请求回传的原来的页面。可以在 Page 类的 Page_Load 事件中使用该属性，以便数据访问代码只在首次加载页面时运行，具体使用会在任务 4-2-1 中进行展示。

4.1.3　Request 对象

1. Request 对象简介

Request 对象派生自 HttpRequest 类，是 HttpRequest 类的一个实例，是 Page 类的成员。它能够读取客户端在 Web 请求期间发送的 HTTP 值，包括从 HTML 表单中用 Post 或者 Get 方法传递的参数、Cookie 和用户认证。在程序中无须做任何声明即可直接使用。它与后边要讲解的 Response 对象一起使用，达到沟通客户端与服务器端的作用，使它们之间可以简单地交换数据，由此可见该对象的重要性。Request 对象最大的用途在于可以接收客户端通过表单提交或 URL 地址串发送过来的信息，同时，也可以获取页面间传递的值，客户端浏览器的信息，客户端的 IP 地址以及当前页面的路径等环境变量。总而言之，所有从前端浏览器通过 HTTP 协议送往后端 Web 服务器的数据，都是借助 Request 对象完成的。其使用语法格式如下：

```
Request . [属性|方法] [变量或字符串]
```

例如：

```
Request.QueryString["user_name"]
```

2. Request 对象的属性和方法

要掌握 Request 对象的使用，必须了解它的常用属性和方法。Request 对象的常用属性和方法如表 4-5 所示。

表 4-5　Request 对象的常用属性和方法

属性和方法	说　明
ApplicationPath 属性	获取目前正在执行程序的服务器的虚拟根路径
Browser 属性	获取有关正在请求的客户端的浏览器功能的信息
Cookie 属性	获取客户端发送的 Cookie 集合
FilePath 属性	获取当前请求的虚拟路径
Files 属性	获取客户端上传的文件集合
Form 属性	获取窗体变量集合
Item 属性	获取 Cookie、Form、QueryString、ServerVariables 集合中指定的对象
Path 属性	获取当前请求的虚拟路径
PhysicalPath 属性	获取当前请求网页在服务器端的物理路径
QueryString 属性	获取 HTTP 查询字符串变量集合
ServerVariables 属性	获取 Web 服务器变量的集合
Url 属性	获取有关目前请求的 URL 信息
UserHostAddress 属性	获取远方客户端机器的主机 IP 地址
MapPath 方法	映射指定的虚拟路径到物理路径
SaveAs 方法	保持 HTTP 请求到硬盘
BinaryRead 方法	以二进制方式读取指定字节的输入流

虽然 Request 对象的属性很多，但常用的只有 QueryString、Path、Browser、UserHostAddress 等。

1）利用 QueryString 集合传递参数

QueryString 属性可以获取标识在 URL 后面的所有返回的变量及其值。在超链接中，常常需要从一个页面跳转到另一个页面，跳转的页面需要获取 HTTP 的值来进行相应的操作，此时，可以在服务器端通过 Request 对象的 QueryString 属性来获取 HTTP 查询字符串变量集合。

通常在以下 3 种情况下使用 QueryString 集合：

（1）当用户将表单中数据提交给服务器时，若将表单的 method 属性设置为 get，则可通过 QueryString 集合来获取客户端所传来的信息（get 方法一般只能传递 256B 的数据，而 post 方法可以达到 2MB）。例如，获取变量 txtName 和 txtSex 所提交的值的代码应为：

```
Label1.Text = Request.QueryString ["txtName"];
Label2.Text = Request.QueryString ["txtSex"];
```

（2）使用 A 标记创建超文本链接时，可以将查询字符串放在 URL 后面，并使用问号“?”来分隔 URL 与查询字符串。例如，单击下面的超链接文本时，将发送一个名为 UserName 的变量，其值为"Tom"，可以通过 QueryString 集合检索查询字符串中变量的值。

```
<a href = "Default.aspx?UserName = Tom">单击这里</a>
```

也可以通过查询字符串发送多个变量，此时要使用“&”符号分隔各个变量，如：

```
<a href = "default.aspx?UserName = Tom&UserAge = 19">单击这里</a>
```

（3）在浏览器地址栏中输入请求网页的 URL 时，可以在 URL 后面输入问号“?”和查询字符串，可以通过 QueryString 集合检索查询字符串中变量的值。如，当客户端送出如下请求时，QueryString 将会得到 Id 和 name 两个变量的值：

```
http://....../ShowPage.aspx?Id = 2&Name = Zhangsan
```

引用上述两个变量的值，可以使用如下方法：

```
Id = Request.QueryString["Id"];
Name = Request.QueryString["Name"];
```

结果为 Id="2"，Name=" Zhangsan "，接收到的数据类型为字符串型。

注意：问号？后面可以有多个变量参数，参数之间用 & 连接。

2）利用 Form 集合接收表单数据

用来获取客户端通过 post 方式提交的数据，相比在客户端用 get 方式传送，post 方式传送的值在地址栏中看不到，安全性较高，但效率较低，可以传送的数据大小没有限制，示例代码如下：

```
Request.Form["txtName"]
```

表示获取表单中名为 txtName 控件的值。

3）使用 UserHostAddress 属性

通过使用 UserHostAddress 的方法，可以获取远程客户端 IP 主机的地址，示例代码如下：

```
Request.UserHostAddress;
```

在客户端主机 IP 统计和判断中，可以使用 Request.UserHostAddress;进行 IP 统计和判断，在有些系统中，需要对来访的 IP 进行筛选，使用 Request.UserHostAddress;就能够轻松地判断用户 IP 并进行筛选操作。

4.1.4　Response 对象

1．Response 对象简介

Response 对象由 System.Web.HttpResponse 类实现，封装了 Web 服务器对客户端请求的响应，它用来操作 HTTP 相应的信息，负责将数据从服务器发送回浏览器。它允许将数据作为请求的结果发送到客户浏览器中，并提供有关响应的信息。它可用来在页面中输入数据、在页面中跳转，还可以传递各个页面的参数。

2．Response 对象的属性和方法

Response 对象的常用属性和方法如表 4-6 所示。

表 4-6　Response 对象的常用属性和方法

属性和方法	说　明
BufferOutput 属性	获得或设置一个值，该值指示是否缓冲输出，并在完成处理整个响应之后将其发送
Charset 属性	获取或设置输出流(文件)的 HTTP 字符集
ContentType 属性	指定送出文件的 MIME 类型。默认文件类型为“text/HTML”，还有“text/GIF”、“text/JPEG”
Cookie 属性	获得响应的 Cookie 集合
Write 方法	用于将信息写入 HTTP 响应输出流，输出到客户端显示
WriteFile 方法	将页面以文件流的方式输出到客户端，常与 Response 对象的 ContentType 属性一起使用
Clear 方法	将缓冲区中的所有 HTML 页面内容清除 语法：Response.Clear()； 此时，Response 对象的 BufferOutput 属性必须设置为 true，否则会报错
Close 方法	关闭客户端的联机
End 方法	将目前缓冲区中的所有 HTML 数据发送到客户端，停止该页的执行，并引发 EndRequest 事件

续表

属性和方法	说　明
Flush 方法	立即将缓冲区中的所有 HTML 数据送到客户端,但不停止页面程序的执行 语法：Response.End() 此时,Response 对象的 BufferOutput 属性必须设置为 true,否则会报错
Redirect 方法	将客户端重定向到新的 URL
BinaryWrite 方法	将一个二进制的字符串写入 HTTP 输出流
AppendToLog 方法	将自定义日志信息添加到 IIS 的日志文件中

1）利用 Write 方法输出信息

利用 Write 方法就可以在客户端输出信息,语法为：

```
Response .Write(变量数据或字符串)
```

例如：

```
Response .Write (user_name&"您好")           //user_name 是一个变量,表示用户名
Response .Write ("欢迎您的光临<p>")           //输出字符串
Response .Write ("现在是北京时间: "&now())   //now()是时间函数
```

2）输出缓存数据

BufferOutput 属性用来设置页面中是否使用缓存技术。页面缓存技术就是页面下载到客户端之前,先暂时存放在服务器端的缓冲区,待页面程序全部编译成功后,再从缓冲区输出到客户端浏览器,这样可以加快用户浏览页面的速度。如果不使用页面缓存技术,页面将一边编译一边向客户端浏览器传送数据,当页面下载量过大时,经常会出现页面不能显示的情况。BufferOutput 属性的取值为 True 或 False,默认为 True,其语法格式如下：

```
Response . BufferOutput = True | False
```

3）输出缓存数据

输出缓存数据就是不等页面完全编译存储到缓冲区,就可以中途将缓存数据输出。如果页面的数据太大,就需要中途将缓存的数据输出,清空缓存,以方便页面继续存到缓存区中。

Response 对象可通过 Flush、End 方法将缓冲区中的数据输出显示到客户端,但 Flush 方法没有停止页面程序的执行,而 End 方法则会停止页面程序的执行。

【示例 4-1】 Response 对象的 Flush 和 End 方法使用比较。

(1) 在 ch04 网站项目中创建文件 FlushEnd.aspx。

(2) 在 FlushEnd.aspx.cs 文件的 Page_Load 方法内添加如表 4-7 所示的代码。

表 4-7　页面 FlushEnd.aspx 的 Page_Load 事件过程的代码

行号	代　码　页	
01	Response .Write ("这是第一句 ");	//输出第一句
02	Response.Flush();	//执行 Flush 方法
03	Response .Write ("这是第二句 ");	//输出第二句

续表

行号	代 码 页	
04	Response.End();	//执行 End 方法
05	Response.Write("这是第三句< br >");	//输出第三句
06	Response.Flush();	
07	Response.Write("这是第四句< br >");	

(3) 浏览该页面，结果如图 4-1 所示。



图 4-1 Response.End 方法示例

在该示例中，第二行代码执行之后，第三行代码还可以执行，输出“这是第二句”；执行第四行代码之后，却没有输出“这是第三句”，这说明 Flush 方法没有终止后面程序的执行，而执行 End 方法之后，终止了后面代码的执行。

4) 使用 Redirect 方法引导客户至另一个 URL 位置

在网页中，可以利用超链接引导客户至另一个页面，但是必须在客户端单击超链接才行。使用 Redirect 方法就可以自动引导客户至另一个页面，亦即重定向，使用该方法时只要传入一个字符串的 URL 即可，其语法格式如下：

```
Response.Redirect(URL)                    //将网页转移到指定的 URL
```

例如：

```
Response . Redirect ("http://www.edu.cn")      //引导至中国教育网
Response . Redirect ("Default.aspx")           //引导至网站内的另一个页面 Default.aspx
```

5) WriteFile 方法

Response 对象的 WriteFile 方法与 Write 方法一样，都是向客户端输出数据。Write 方法是输出这个方法中带的字符串，而 WriteFile 方法则可以输出二进制信息，它不进行任何字符转换，直接输出。其语法格式为：

```
Response.WriteFile (变量或字符串)
```

如下面的例子将显示一张图片：

```
Response.ContentType = "image/JPEG";          //定义文件类型
Response.WriteFile ("logo.jpg");              //输出图片文件
```

4.1.5 Cookie 对象

1. Cookie 对象简介

Cookie 对象是 System.Web 命名空间中 HttpCookie 类的对象，是一种可以在客户端保存信息的方法。使用 Cookie 对象能够持久化的保存用户信息，所以 Cookie 对象能够长期保存。当用户访问某个的站点时，该站点可以利用 Cookie 保存用户首选项或其他信息，这样当用户下次再访问该的站点时，Web 应用程序可以通过获取客户端的 Cookie 信息来判断用户的身份进行认证。

用户每次访问站点时，Web 应用程序发送给该用户一个页面和一个包含日期和时间的 Cookie。用户的浏览器在获得页面的同时也获得了该 Cookie，并且将它存储在用户本地磁盘中。以后如果用户再次请求该站点中的页面，浏览器就会在用户本地硬盘上查找与该网站相关联的 Cookie。比如当用户登录某些网站的邮箱后，如果在 Cookie 中记录了用户名信息，那么在 Cookie 信息失效之前，该用户在同一台计算机再次登录时就不需要提供用户名了。

由于 HTTP 协议是一个无状态的协议，所以，对于页面的每一次请求，都被看作是一次新的会话。这样就无法知道用户最近都访问了哪些页面，这对于那些需要获取用户身份才能工作下去的应用来说十分不方便。而 Cookie 作为用户和服务器之间进行交换的小段信息，弥补了 HTTP 协议的这一缺陷。

Cookie 中保存的信息片断以“键/值”对的形式存储，一个“键/值”对仅仅是一条命名的数据。一个网站只能取得它放在用户的计算机中的信息，它无法从其他的 Cookie 文件中取得信息，也无法得到用户的计算机上的其他任何东西。

Cookie 有两种形式：会话 Cookie 和永久 Cookie。会话 Cookie 是临时性的，只有打开浏览器时才存在，一旦会话结束或超时，这个 Cookie 就不存在了。永久 Cookie 则是永久性地存储在用户的硬盘上，并在指定的日期之前一直可用。相比于后面将要介绍的 Session 和 Application 而言，使用 Cookie 的优点可以归纳如下几点。

- 可配置到期规则。Cookie 可以在浏览器会话结束时到期，或者可以在客户端计算机上无限期存在。
- 不需要任何服务器资源。Cookie 存储在客户端并在发送后由服务器读取。
- 简单性。Cookie 是一种基于文本的轻量结构，包含简单的键值对。
- 数据持久性。Cookie 通常是客户端上持续时间最长的数据保留形式。

虽然 Cookie 具有若干优点，这些优点能够弥补 Session 对象和 Application 对象的不足，但是 Cookie 对象同样有缺点，读者可以查阅相关资料，在此不予赘述。

2. Cookie 对象的属性和方法

Cookie 对象的主要属性如下：

- Domain——获取或设置将此 Cookie 与其关联的域。
- Expires——获取或设置此 Cookie 的过期日期和时间。

- Name——获取或设置 Cookie 的名称。
- Path——获取或设置输出流的 HTTP 字符集。
- Secure——获取或设置一个值，该值指示是否通过 SSL（即仅通过 HTTPS）传输 Cookie。
- Value——获取或设置单个 Cookie 值。
- Values——获取在单个 Cookie 对象中包含的键值对的集合。

Cookie 对象的主要方法如下：

- Add——添加一个 Cookie 变量。
- Clear——清除 Cookie 集合中的变量。
- Get——通过索引或变量名得到 Cookie 变量值。
- GetKey——以索引值获取 Cookie 变量名称。
- Remove——通过 Cookie 变量名称来删除 Cookie 变量。

3. Cookie 对象的使用

对象 Request 和 Response 都提供了一个 Cookie 集合。可以利用 Response 对象设置 Cookie 的信息，而使用 Request 对象获取 Cookie 的信息。

注意：Cookie 通过 Response 对象发送到浏览器进行创建时，需要指定 Name 属性和 Value 属性。每个 Cookie 必须有一个唯一的名称，以便以后从浏览器读取 Cookie 时可以识别它。由于 Cookie 按名称存储，因此用相同的名称命名两个 Cookie 会导致其中一个 Cookie 被覆盖。

有两种方法可以向用户计算机写入 Cookie。可以直接为 Cookie 集合设置 Cookie 属性，也可以创建 HttpCookie 对象的一个实例并将该实例添加到 Cookie 集合中。下面的代码演示了两种编写 Cookie 的方法：

```
Response.Cookie["userName"].Value = "Sina";
Response.Cookie["userName"].Expires = DataTime.Now.AddDays(2);
```

或

```
HttpCookie myCookie = new HttpCookie("myCookie");
myCookie.Value  =  "sina";                          //设置 Cookie 的值
myCookie.Expires  =  System.DateTime.Now.AddDays(30);    //设置过期时间为 30 天
Response.Cookie.Add(myCookie);
```

上述两段代码演示向 Cookie 集合添加了两个 Cookie，一个名为 userName，另一个名为 myCookie。对于第一个 Cookie，Cookie 集合的值是直接设置的。对于第二个 Cookie，代码创建了一个 HttpCookie 类型的对象实例，设置其属性，然后通过 Add 方法将其添加到 Cookie 集合中。在实例化 HttpCookie 对象时，必须将该 Cookie 的名称作为构造函数的一部分进行传递。

浏览器向站点发出请求时，会随请求一起发送该站点的 Cookie，可以用 Request 对象读

取 Cookie，并且读取方式与将 Cookie 写入 Response 对象的方式基本相同。下面的代码演示了两种方法，通过这两种方法可以获取名为 userName 的 Cookie 的值，并将其赋值给一个字符串变量 cookieName：

```
if (Request.Cookie["userName"]!= null)
  string cookieName = Request.Cookie["userName"].Value;
```

或

```
if (Request.Cookie["userName"]!= null)
{
    HttpCookie myCookie = Request.Cookie["userName"];
    string cookieName = myCookie.Value;
}
```

在尝试获取 Cookie 的值之前，应确保该 Cookie 存在；如果该 Cookie 不存在，将会产生异常。

【示例 4-2】 Cookie 对象使用示例。

(1) 在 ch04 网站项目中创建 CookieDemo.aspx 页面

(2) 在 CookieDemo.aspx.cs 文件的 Page_Load 方法内添加如表 4-8 所示的代码，创建一个 Cookie，当下次客户登录时获取到上次写入的 Cookie 信息。

(3) 用户第一次登录网站时，结果如图 4-2 所示，获取 Cookie 信息时出错，抛出异常错误，并执行第一次写入，当下一次运行或者刷新页面时，将看到如图 4-3 所示的结果，将上一次写入到 Cookie 的信息读出并显示。

表 4-8 页面 CookieDemo.aspx 的 Page_Load 事件过程的代码

行号　代　码　页

```
try
{
    HttpCookie myCookie = new HttpCookie("Sina ");                        //创建 Cookie 对象
    myCookie.Value = Server.HtmlEncode("一个 Cookie 应用程序");          //Cookie 赋值
    myCookie.Expires = DateTime.Now.AddDays(5);                          //Cookie 持续时间
    Response.AppendCookie(myCookie);                                     //添加 Cookie
    Response.Write("Cookie 创建成功");                                   //输出成功
    Response.Write("<hr/>获取 Cookie 的值<hr/>");
    HttpCookie GetCookie = Request.Cookie["myCookie"];                   //获取 Cookie
    Response.Write("Cookie 的值:" + GetCookie.Value.ToString() + "<br/>");  //输出 Cookie 值
    Response.Write("当前时间: " + DateTime.Now.ToString() + "<br/>");
    Response.Write("Cookie 的过期时间:" + myCookie.Expires.ToString() + "<br/>");
}
catch
{
    Response.Write("获取 Cookie 信息失败");                              //抛出异常
}
```



图 4-2　获取不存在的 Cookie 信息



图 4-3　获取存在的 Cookie 信息

4.1.6　Session 对象

1. Session 对象简介

在上网时，可以利用超链接方便地从一个页面跳转到另一个页面。但是怎样记载客户的信息呢？例如，用户在首页输入了自己的用户名和密码，如果在其他页面还要使用该用户名，那么如何记住用户在首页输入的用户名呢？

迄今为止，我们能用两种方法解决：

- 利用 Request 对象的 QueryString 方法一页一页地传过去，这种方法太麻烦。
- 利用 Cookie 保存用户名。

下面学习一种更加简洁的方法：利用 Session(会话)对象。

Session 对象是由 System. Web 命名空间中的 HttpSessionState 类实现的，用来记载特定客户的信息。即使该客户从一个页面跳转到另一个页面，该 Session 信息仍然存在，客户在该网站的任何一个页面都可以存取 Session 信息。Session 对象变量只针对单一网页的使用者，也就是说，访问同一站点的不同用户之间的 Session 对象不尽相同，即每个 Session 对象中的信息只能被用户自己使用，而不能被网站的其他用户访问，因此可以在不同的页面间共享数据(如上述的在首页中输入的用户名)，但是不能在用户间共享数据。

注意：Session 信息是针对一个客户而言的，不同客户的信息用不同的 Session 对象记载。例如，用户 A 和用户 B，均访问同一个 Web 网站，当用户 A 访问时，该站点显示地为用户 A 增加一个 Session 值，同样，当用户 B 访问该站点时，又为用户 B 增加一个 Session 值。

Session 的工作原理还是比较复杂的：ASP.NET 采用一个具有 120 位的标识符来跟踪每一个 Session。ASP.NET 中利用专有算法来生成这个标识符的值，从而保证了(统计上的)这个值是独一无二的，这个特殊的标识符就被称为 SessionID，当用户第一次访问一个网站时，服务器就给该用户建立了一个 Session 对象，并分配一个唯一的 SessionID。SessionID 是传播于网络服务器和客户端之间的唯一的一个信息。当客户端出示它的 SessionID，ASP.NET 找到相应的 Session，从状态服务器里获得相应的序列化数据信息，从而激活该 Session，并把它放到一个可以被程序访问的集合里。为使系统能够正常工作，客户端必须为每个请求保存相应的 SessionID，获取某个请求的 SessionID 的方式有两种：

- 使用 Cookie。在这种情况下，当 Session 集合被使用时，SessionID 被 ASP.NET 自动转化一个特定的 Cookie(被命名为 ASP.NET_SessionID)。

- 使用改装的 URL。在这种情况下，SessionID 被转化为一个特定的改装的 URL。ASP.NET 的这个新特性可以让程序员在客户端禁用 Cookie 时创建 Session。

2. Session 对象的属性和方法

Session 对象的属性主要有：

- SessionID——对于不同的用户会话，SessionID 是唯一的，只读属性。
- Timeout——Session 的有效期时长，即一个会话结束之前会等待用户没有任何活动的最长时间，默认为 20 分钟。
- Keys——根据索引号获取变量值。
- Count——获取会话状态集合中的项数。

Session 对象的方法主要有：

- Abandon——清除用户的 Session 对象，释放系统资源。
- Add——添加一个新项到会话状态中。
- Clear——清除当前会话状态的所有值。
- Remove——删除会话状态集合中的项。
- RemoveAt——删除会话状态集合中指定索引处的项。
- RemoveAll——清除会话状态集合中的所有的键和值。

3. Session 对象的使用

利用 Session 存储信息其实很简单，可以把变量或字符串等信息很容易地保存在 Session 中。语法如下：

```
Session ["Session 名字"] = 变量、常量、字符串或表达式
```

例如：

```
Session ["user_name"] = name //name 为变量
Session ["age"] = 18
Session ["school"] = "北京大学"
```

4. Session_Start 和 Session_End 事件

Session_Start 事件在 Session 对象开始时被触发。通过 Session_Start 事件可以统计应用程序当前访问的人数，同时也可以进行一些与用户配置相关的初始化工作。

```
protected void Session_Start(object sender, EventArgs e)
{
    Application ["online"] = Application ["online"] + 1;     //在线人数加 1
}
```

与之相反的是 Session_End 事件，当 Session 对象结束时被触发。当通过 Session 对象统计应用程序当前访问在线人数时，可以通过 Session_End 时间减少在线人数统计数字，同

时也可以对用户配置进行相关的清理工作。

```
protected void Session_End(object sender, EventArgs e)
{
    Application ["online"] = Application ["online"] - 1;    //在线人数减 1
}
```

4.1.7　Application 对象

1. Application 对象简介

Application 对象是 System. Web 命名空间中的 HttpApplicationState 类的实例，用来保存所有客户的公共信息。Application 对象为经常使用的信息提供了一个有用的 Web 站点存储位置，Application 中的信息可以被网站的所有页面访问，因此可以在不同的用户间共享数据。

注意：Application 对象变量是建立在内存中的，这个状态变量可以被网站的所有用户访问。

Application 对象内保存的信息可以在 Web 服务整个运行期间保存，是应用程序级的对象，用来存储 ASP.NET 应用程序中多个会话和请求之间的全局共享信息，与此相反，Session 对象可以记载特定客户的信息。简言之，不同的客户可以访问公共的 Application 对象，但必须访问不同的 Session 对象。

Application 对象具有如下特点：

- 数据可以在 Application 对象内部共享。
- 一个 Application 对象包含事件，可以触发某些 Application 对象脚本。
- 个别 Application 对象可以用 Internet Service Manager 来设置而获得不同属性。
- 单独的 Application 对象可以隔离出来在它们自己的内存中运行。
- 可以停止一个 Application 对象(将其所有组件从内存中驱除)而不会影响到其他应用程序。
- 一个网站可以有不止一个 Application 对象。典型情况下，可以针对个别任务的一些文件创建个别的 Application 对象。
- Application 对象成员在服务器运行期间持久地保存数据。Application 对象成员的生命周期止于关闭 IIS 或使用 Clear 方法清除。
- 因为多个用户可以共享一个 Application 对象，所以必须要有 Lock 和 Unlock 方法，以确保多个用户无法同时改变某一属性。

2. Application 对象的属性和方法

Application 对象的属性主要有：

- AllKeys——获取 HttpApplicationState 集合中的访问键。
- Contents——获取对 HttpApplicationState 对象的引用。
- Count——获取 HttpApplicationState 集合中的对象数。

- Count——通过名称和索引获取 HttpApplicationState 集合中的对象数。
- Keys——获取 NameObjectCollectionBase. KeysCollection 实例，该实例包含 NameObjectCollectionBase 实例中的所有键。

Application 对象的方法主要有：

- Add——新增一个 Application 对象变量。
- Clear——清除全部 Application 对象变量。
- Remove——按变量名称移除一个 Application 对象变量。
- Lock——锁定 Application 对象。
- UnLock——解除对 Application 对象的锁定。

3. 利用 Application 对象存储信息

Application 的使用方法和 Session 非常类似，可以把变量、字符串等信息很容易地保存在 Application 中。语法格式如下：

```
Application ["Application 名字"] = 变量、常量、字符串或表达式
```

可以直接把变量、字符串等信息保存在 Application 中，当 Web 应用不希望用户在客户端修改已经存在的 Application 对象时，可以使用 Lock 对象进行锁定，当执行完相应代码块后可以解锁，代码如下所示。

```
Application .Lock()
Application ["user_name"] = user_num          //将 user_num 变量存入 Application
Application ["city"] = "南京"                 //将字符串信息存入 Application
Application .Unlock()
```

注意：Lock 方法和 Unlock 方法是很重要的，因为任何客户都可以存取 Application 对象，如果正好有两个客户同时更改一个 Application 对象的值怎么办？可以利用 Lock 方法先将 Application 对象锁定。以防止其他客户更改。更改后，再利用 Unlock 方法解除锁定。不过，读取 Application 对象时就没必要这样做了。

4. Global. asax 文件

Global. asax 文件(也称为 ASP.NET 应用程序文件)是一个可选的文件，该文件包含响应 ASP.NET 或 HTTP 模块所引发的应用程序级别和会话级别事件的代码。Global. asax 文件驻留在 ASP.NET 应用程序的根目录中。

每个 ASP.NET 应用程序都可以有一个 Global. asax 文件。一旦将其放在适当的虚拟目录中 ASP.NET 就会把它识别出来并且会自动使用该文件。

注意：Global. asax 存放的位置是固定的，必须存放在当前应用程序所在的虚拟根目录下，如果存在虚拟目录的子目录中，Global. asax 文件将不会起任何作用。

1）创建 Global. asax 配置文件

Global. asax 配置文件通常处理高级的应用程序事件，如 Application _ Start、Application_End、Session_Start 等，创建 Global. asax 配置文件可以通过新建“全局应用程

序类”文件来创建，如图 4-4 所示。

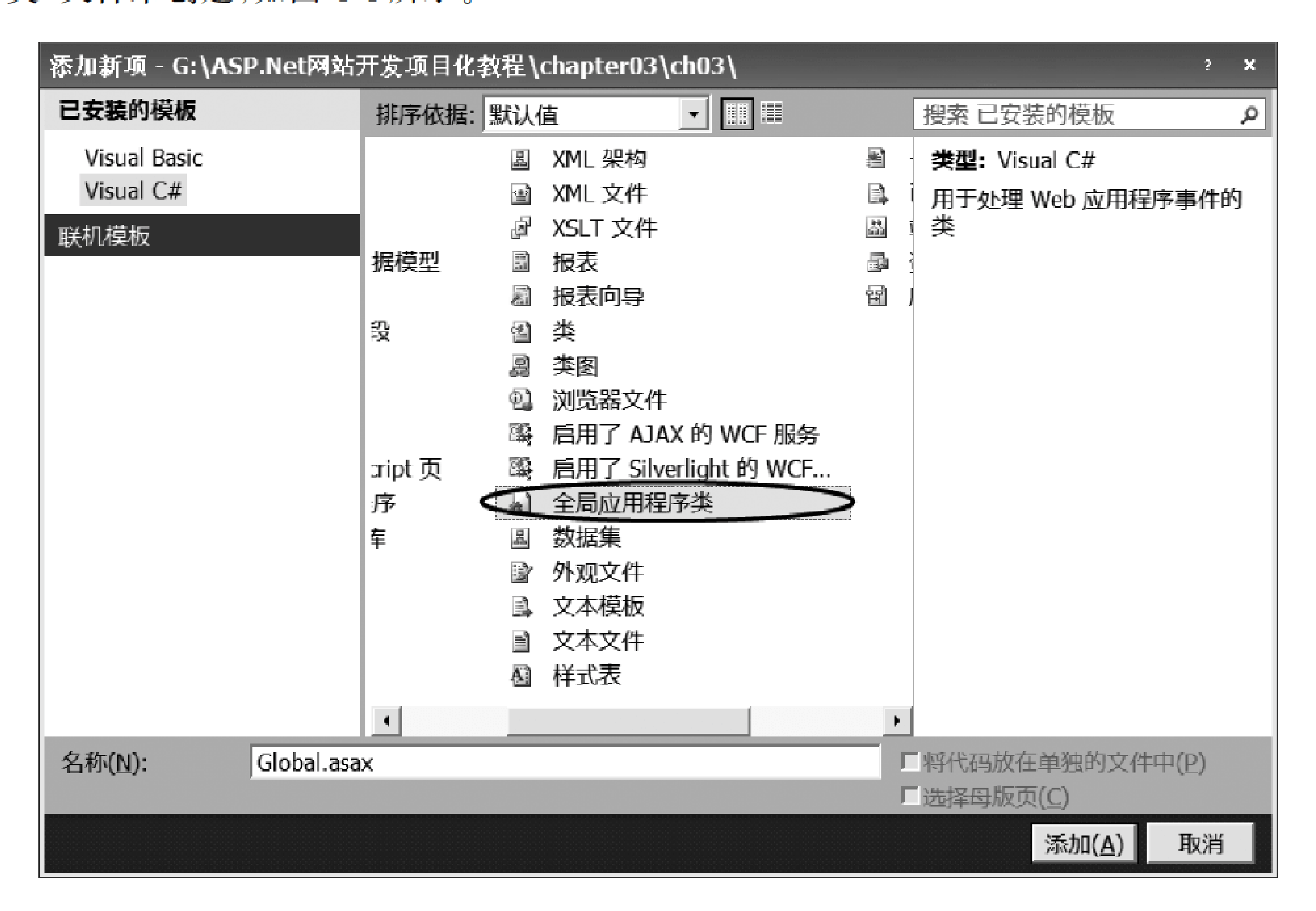


图 4-4　创建 Global. asax 配置文件

2）Application_Start 和 Application_End 事件

在 Global. asax 配置文件中，Application_Start 事件会在 Application 对象被创建时触发，通常，Application_Start 事件能够对应用程序进行全局配置。例如在统计网站在线人数时，通过重写 Application_Start 方法可以实现实时在线人数统计，可以在 Global. asax 文件的 Application_Start 方法中添加如下代码：

```
void Application_Start(object sender, EventArgs e)
{
    //在应用程序启动时运行的代码
    Application["count"] = 0;      //创建 Application 对象
}
```

与之相反，当用户离开当前 Web 应用时，就会触发 Application_End 事件，可以在 Application_End 方法中清理相应的用户数据。

注意：Session 对象和 Application 对象都能够进行应用程序中在线人数或应用程序的统计和计算，在选择对象时，可以根据具体应用要求，选择不同的内置对象。

4.1.8　Server 对象

1. Server 对象简介

Server 对象由 System . Web. HttpServerUtility 类实现，它提供了访问服务器的一些非

常有用的属性和方法，Server 对象是专门为处理服务器上特定任务而设计的，特别是与服务器的环境和处理活动有关的任务，主要用于创建 COM 对象和 Scripting 组件、转化数据格式、管理其他页的执行。语法格式为：

```
Server.方法(变量或字符串)
Server.属性 = 属性值
```

Server 对象的属性主要有：

- MachineName——获取远程服务器的名称。
- ScriptTimeout——获取和设置请求超时。

Server 对象的方法主要有：

- CreatObject——创建 COM 对象的一个服务器实例。
- Execute——使用另一个页面执行当前请求。
- Transfer——终止当前页面的执行，并为当前请求开始执行新页面。
- HtmlDecode——对已被编码的消除 Html 无效字符的字符串进行解码。
- HtmlEncode——对要在浏览器中显示的字符串进行编码。
- MapPath——返回与 Web 服务器上的执行虚拟路径相对应的物理文件路径。
- UrlDecode——对字符串进行解码，该字符串为了进行 HTTP 传输而进行编码并在 URL 中发送到服务器。
- UrlEncode——编码字符串，以便通过 URL 从 Web 服务器到客户端浏览器的字符串传输。

2. Server 对象的使用

1）使用 MachineName 属性获取服务名称

在网站项目下创建页面 MachineNameDemo.aspx，并添加如下代码。

```
服务器名称: <asp:Label ID = "lbMnInfo" runat = "server" Text = "Label" ForeColor = "Red">
</asp:Label>
```

在页面后置代码文件 MachineNameDemo.aspx.cs 中添加以下代码获取服务器名称，并将服务器名称变成小写字母形式输出。

```
protected void Page_Load(object sender, EventArgs e)
{
    this.lbMnInfo.Text = Server.MachineName.ToLower();
}
```

运行页面 MachineNameDemo.aspx，效果如图 4-5 所示。

2）使用 ScriptTimeout 属性设置超时时间

该属性用来规定脚本文件执行的最长时间，默认为 90 秒。如果超出最长时间脚本文件还没有执行完，就自动停止执行。这样可以防止某些可能进入死循环的错误导致服务器过载问题。

```
Server.ScriptTimeout = 300;          //将最长执行时间设置为 300 秒
```

3）用 Execute 方法执行对另一页的请求

该方法用来停止执行当前网页，转到新的网页执行，执行完后返回原网页，继续执行 Execute 方法后面的语句。语法格式如下：



```
Server . Execute (变量或字符串);
```



图 4-5　使用 MachineName 属性获取服务名称

例如：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("<p>调用 Execute 方法之前</p>");
    Server.Execute("TestPage.aspx");    //该页 Page_Load 有 Response.Write("测试");语句
    Response.Write("<p>调用 Excute 方法之后</p>");
}
```

4）用 Tranfer 方法实现网页重定向

该方法和 Execute 方法非常相似，唯一的区别是执行完新的网页后，并不返回原网页，而是停止执行过程。该方法可以把控制传递出去，可以把原来页面的所有内置对象和这些对象的状态都传递给新的页面，比如 Request 对象的查询字符串。使用这种方法还可以把一个大的程序划分成小的模块，然后用 Tranfer 方法把各个模块联系起来，语法格式如下：

```
Server . Transfer (变量或字符串)
```

例如：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("<p>调用 Transfer 方法之前</p>");
    Server.Tranfer("TestPage.aspx");    //该页 Page_Load 有 Response.Write("测试");语句
    Response.Write("<p>调用 Transfer 方法之后</p>");
}
```

5）用 MapPath 方法将虚拟路径转化为实际路径

在创建文件、删除文件或者读取文件类型的数据库时（如 Access 和 SQLite），都需要指定文件的路径并显式地提供物理路径执行文件的操作，如 D:\Program Files。但是这样做却暴露了物理路径，如果有非法用户进行非法操作，很容易就显示了物理路径，这样就造成了安全问题。

在使用文件和创建文件时，如果非要为创建文件的保存地址设置一个物理路径，这样非常不便并且用户体验也不好。当用户需要上传文件时，用户不可能知道也不应该知道服务器路径。而使用 MapPath 方法就能克服以上问题。MapPath 方法以“/”开头，则返回 Web 应用程序的根目录所在的路径，若 MapPath 方法以“../”开头，则会从当前目录开始寻找上

级目录，如图 4-6 所示，而其实际服务器路径如图 4-7 所示。

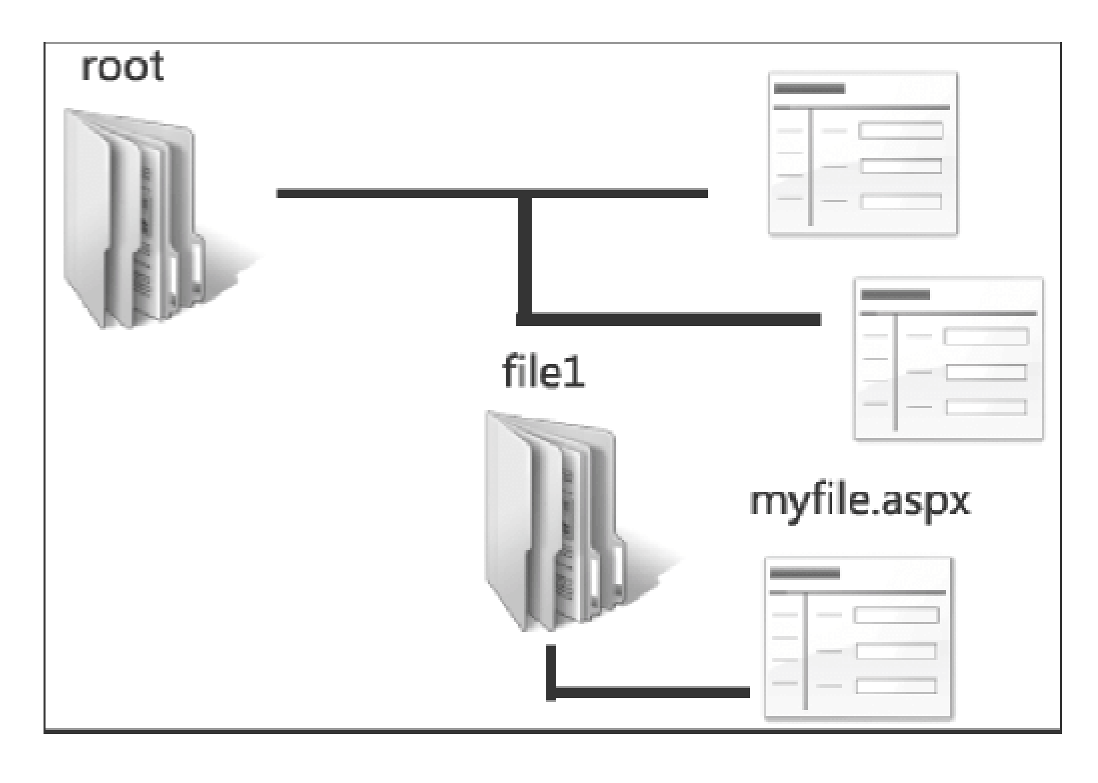


图 4-6 MapPath 示意图



图 4-7 服务器路径

在图 4-6 中，其中根目录为 root，在根目录下有一个文件夹为 file1，在 file1 中的文件使用 MapPath 方法访问根目录中文件的方式有 Server. MapPath（“../文件名称”）或 Server. MapPath（“/文件名称”），示例代码如下所示。

```
string FilePath = Server.MapPath("../Default.aspx");     //设置路径
string FileRootPath = Server.MapPath("/Default.aspx");  //设置路径
```

Server. MapPath 其实返回的是物理路径，但是通过 MapPath 的封装，通过代码无法看见真实的物理路径，若需要知道真实的物理路径，只需输出 Server. MapPath 即可，示例代码如下所示。

```
Response.Write(Server.MapPath("../Default.aspx"));        //输出路径
```

输出结果为 G:\ASP. Net 网站开发项目化教程\chapter04\ch04\ServerDemo. aspx，该结果针对不同的物理路径而不同，结果如图 4-8 所示。



图 4-8 页面 myfile. aspx 运行效果

6）使用 HtmlEncode 和 HtmlDecode 方法进行编码和解码

在 ASP.NET 中，默认编码是 UTF-8，所以在使用 Session 和 Cookie 对象保存中文字符或者其他字符集时经常会出现乱码，为了避免乱码的出现，可以使用 HtmlDecode 和 HtmlEncode 方法进行编码和解码。HtmlEncode 方法用来转化字符串，它可以将字符串中的 HTML 标记转换为字符实体，如将“＜”转换为“<”，将“＞”转换为“>”。其语法格式如下：

```
Server.HtmlEncode（变量或字符串）
Server.HtmlDecode(变量或字符串)
```

特别是 HtmlEncode 方法在需要输出 HTML 语句时非常有用。浏览器是解释执行的，它将网页文件中的 HTML 标记逐一解释执行。但是，有时候就希望直接将 HTML 标记输出到屏幕上，比如在考试 HTML 知识时，就需要在页面中输出 HTML 语句。

另外，还可以通过 HtmlEncode 方法放置脚本入侵。脚本入侵是指网络上一些恶意用户在提交给页面的信息中包括一些特殊脚本程序（如＜script＞＜/script＞），如果没有对其进行特殊处理，则服务器将会这行这些脚本代码。

首先，在网站项目 ch04 中创建页面 HtmlEncodeDemo. aspx，并添加如下代码。

```
<form id="form1" runat="server">
    <div>
        解码前的输出：<asp:Label ID="lbMsg" runat="server" Text="Label"></asp:Label><br />
        使用 HtmlEcode 后的输出：<asp:Label ID="lbHtmlEcode" runat="server"
                                     Text="Label"></asp:Label><br />
        使用 HtmlDecode 后的输出：<asp:Label ID="lbHtmlDecode" runat="server"
                                     Text="Label"></asp:Label>
    </div>
</form>
```

在页面后置文件 HtmlEncodeDemo. aspx. cs 中添加如下代码。

```
protected void Page_Load(object sender, EventArgs e)
{
    String TestString = "测试<H3>方法</H3>";
    lbMsg.Text = TestString;                              //直接输出
    String EncodedString = Server.HtmlEncode(TestString);
    lbHtmlEcode.Text = EncodedString;                     //进行编码转换后的输出效果
    lbHtmlDecode.Text = Server.HtmlDecode(EncodedString); //解码效果
}
```

运行 HtmlEncodeDemo. aspx 页面，效果如图 4-9 所示。

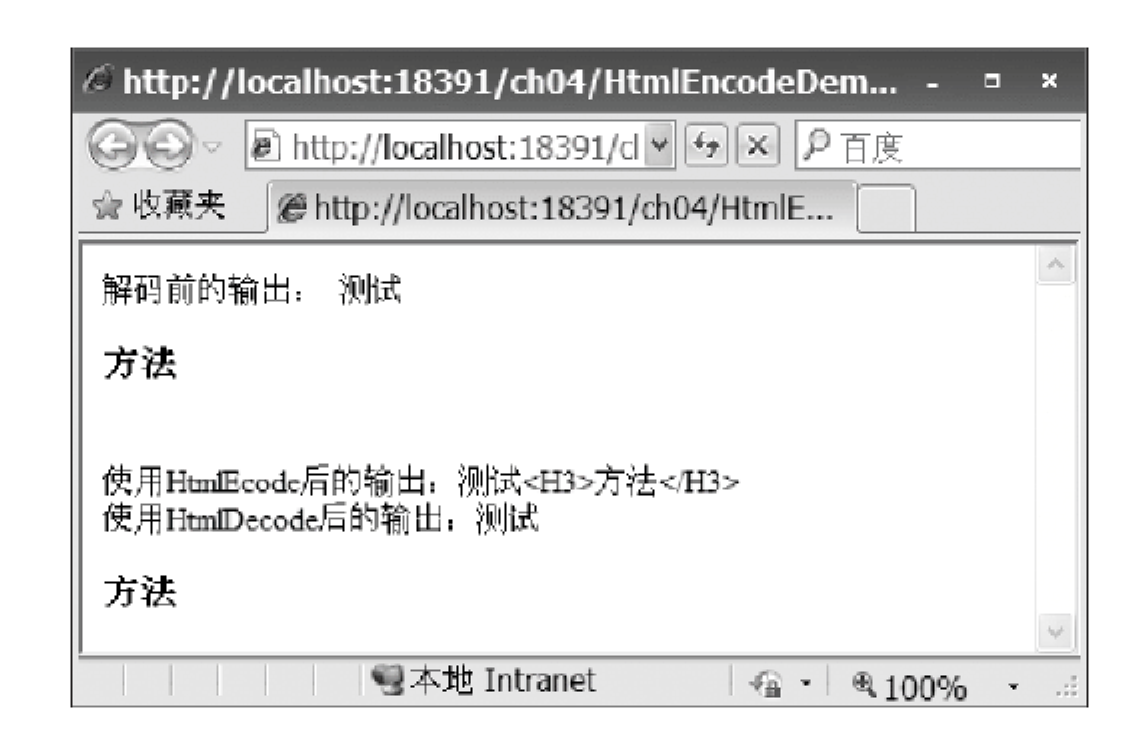


图 4-9　HtmlDecode 和 HtmlEncode 方法示例

7）使用 URLEncode 和 URLDecode 方法进行 URL 编码和解码

该方法也是用来转化字符串的，它可以将其中的特殊字符，像?、&、/和空格等转化为 URL 编码，如把空格转化为它的 URL 编码"+"。语法如下：

```
Server . URLEncode (字符串)
```

为什么要使用该方法呢？主要有以下两个原因：

- 目前的操作系统允许文件名有空格等特殊字符，如果使用 IE 浏览器，一般没有问题，因为浏览器会自动转化空格等字符。但是如果使用别的浏览器，就可能不支持空格等特殊字符。此时就需要使用 URLEncode 方法进行人工转化。
- 在利用 Request 对象的 QueryString 方法获取标识在 URL 后面的参数时，参数可能带有空格等特殊字符，如"<a href="transferArg.aspx? name=张三">"，IE 浏览器一般能正确识别，而其他浏览器可能就无法识别空格以后的字符，从而认为 name 的值是"张"，这时候也需要用 URLEncode 方法进行转化。比如修改为如下代码：

```
<a href = "transferArg.aspx?name = <% = Server.URLEncode("张 三") %>">
```

在页面提交的信息中，由于包括文字、数字、特殊符号等，并以 UTF-8 编码提交到服务器时，经常出现乱码。要解决这一问题就需要通过 Server 对象的 URLEncode 方法对其进行 URL 编码转换，再通过 URLDecode 方法进行解码。URL 编码可以确保所有浏览器均正确地传输 URL 字符串中的文本。URLDecode 方法将 URL 编码向文本字符串进行解码转换。

在网站项目 ch04 中创建页面 URLEncodeDemo.aspx，在该页面设计视图中拖放两个 Button 按钮、两个 Label 和一个 TextBox，代码如下。

```
<form id = "form1" runat = "server">
  <div>
    请输入 URL 进行编码: <asp:TextBox ID = "txtUrl" runat = "server" Width = "258px"></asp:
TextBox>
    <asp:Button ID = "btnUrlEncode" runat = "server" OnClick = "btnUrlEncode_Click"
                Text = "UrlEncode" /><br />
    编码后的 URL 为: <asp:Label ID = "lbUrlEncode" runat = "server" Text = "Label"></asp:
Label><br />
    对编码后的 URL 进行解码: <asp:Button ID = "btnUrlDecode" runat = "server"
                                OnClick = "btnUrlDecode_Click" Text = "UrlDecode" /><br />
    解码后的 URL 为: <asp:Label ID = "lbUrlDecode" runat = "server" Text = "Label"></asp:Label>
  </div>
</form>
```

两个按钮单击事件方法在页面后置文件 URLEncodeDemo.aspx.cs 中的代码如下。

```
protected void btnUrlEncode_Click(object sender, EventArgs e)
{
    this.lbUrlEncode.Text = Server.UrlEncode(txtUrl.Text);
}
```

```
protected void btnUrlDecode_Click(object sender, EventArgs e)
{
    this.lbUrlDecode.Text = Server.UrlDecode(lbUrlEncode.Text);
}
```

运行页面 URLEncodeDemo.aspx,效果如图 4-10 所示。

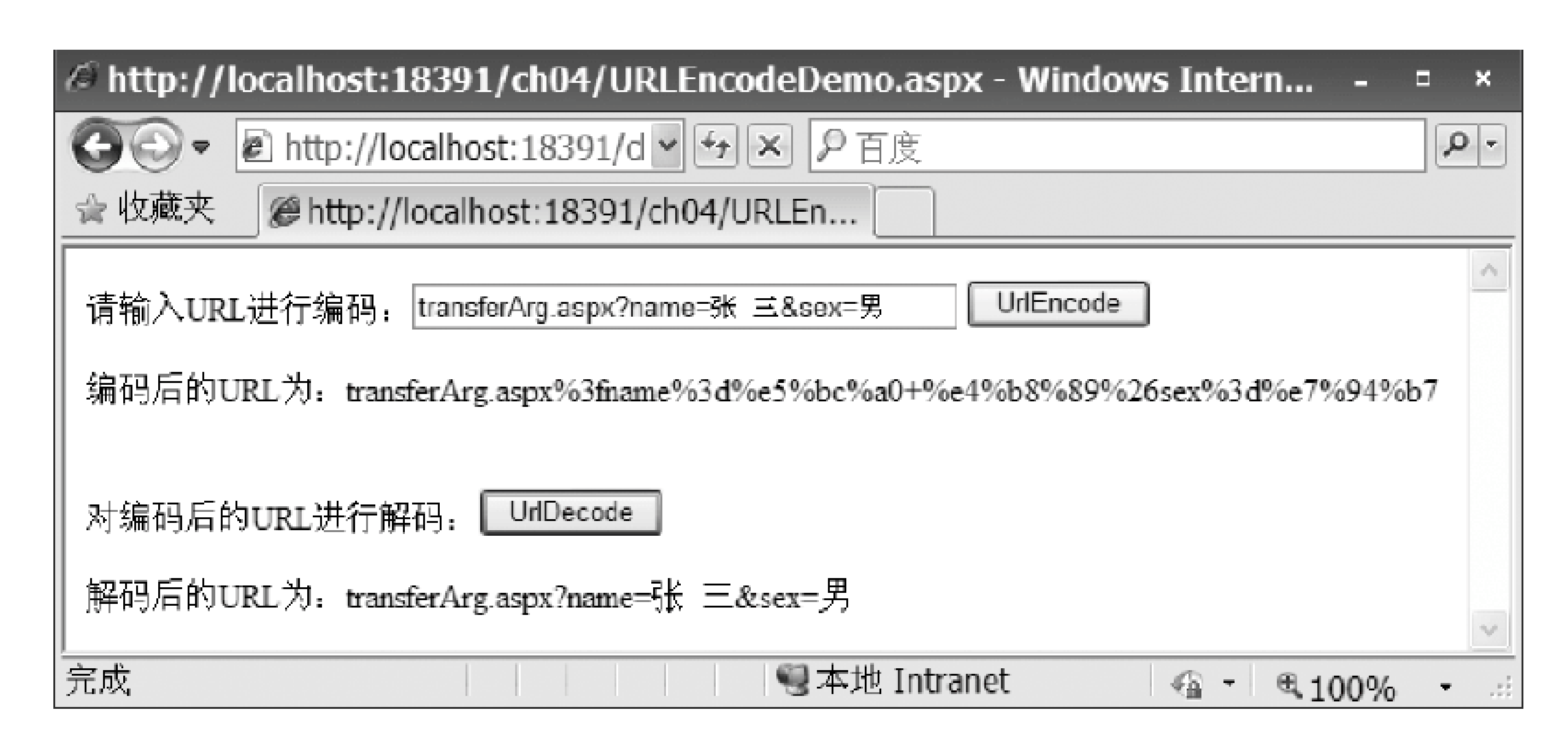


图 4-10　URLEncode 和 URLDecode 方法示例

4.2 单元任务

任务 4-2-1　体验页内数据传递

【任务描述】

将用户在文本框中输入的歌手添加到列表控件中显示,观察添加的结果有什么问题。是怎么引起的?如何解决该问题?

【任务实施】

(1) 创建网站项目 rw4-2-1,并在网站项目下添加页面文件 Default.aspx,进入页面视图,从工具箱分别拖放一个 ListBox、TextBox 和 Button 空间到编辑区,切换至源视图,并编写如下代码。

```
<form id="form1" runat="server">
  <div>
    请选择您的喜欢的歌手<br />
    <asp:ListBox ID="lbSonger" runat="server" Height="151px" Width="152px"></asp:
ListBox>
    <br />
    向列表中添加歌手<br />
    <asp:TextBox ID="txtName" runat="server"></asp:TextBox>
    <asp:Button ID="btnAddSonger" runat="server" Text="添加" OnClick="Button2_Click" />
  </div>
</form>
```

lbSonger 服务器列表控件用于显示歌手的列表，文本框 txtName 用于接收输入。

（2）在页面后置代码文件 Default.aspx.cs 中编写 Page_Load 事件方法代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    lbSonger.Items.Add("帕瓦罗蒂");
    lbSonger.Items.Add("多明戈");
    lbSonger.Items.Add("卡雷拉斯");
}
```

（3）编写“添加”按钮在页面后置代码文件 Default.aspx.cs 中对应的 Click 事件方法代码如下。

```
protected void btnAddSonger_Click(object sender, EventArgs e)
{
    lbSonger.Items.Add(txtName.Text);
}
```

（4）运行页面 Default.aspx，在文本框中输入歌手的名字“韩红”，单击“添加”按钮，发现列表控件中并没有像我们想象的那样只添加了“韩红”，而是重复添加了前面 3 个歌手列表项，如图 4-11 所示。



图 4-11 设置 IsPostBack 属性前的运行结果

导致该问题的原因是在页面加载时没有判断当前页面是否是回传页面，导致每次都要执行 Page_Load 方法中的添加前面 3 个歌手的代码，也就是说，当用户单击“添加”按钮后，页面回传，这段代码有一次被执行。为了解决这个问题，可以使用 ASP.NET 提供的一个特殊属性 Page.IsPostBack 来进行判断，使用过程可以直接写成 IsPostBack，它是一个布尔值，当该值为真(true)时，则页面为回传，否则就是首次加载。

（5）清楚了问题产生的原因后，该问题就很好解决了，修改页面 Default.aspx 的 Page_Load 事件方法代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        lbSonger.Items.Add("帕瓦罗蒂");
        lbSonger.Items.Add("多明戈");
        lbSonger.Items.Add("卡雷拉斯");
    }
}
```

（6）再次运行 Default.aspx 页面，输入歌手名，单击“添加”按钮，就会看到输入的歌手名被添加到列表最后，并且上面的列表项也没有重复添加，如图 4-12 所示。

页内数据传递是最简单的页面数据传递形式，当用户单击页面上的按钮等引起回传的控件时，所有页面上的服务器端控件的值都要回传，而且这些是不需要我们来处理的，ASP.NET 都已经封装好了，所以在 ASP.NET 中可以使用“控件对象名.属性”的方式直接访问控件的相关内容。

图 4-12　设置 IsPostBack 属性后的运行结果

任务 4-2-2　实现简单加法计算器

【任务描述】

编写一个页面，实现当用户每次单击 Button 按钮时，Label 显示的内容自动增 1。

【任务实施】

（1）创建网站项目 rw4-2-2，并在网站项目下添加页面 Default.aspx，从工具箱中拖放一个 Button 控件和一个 Label 至页面，并将 Label 标签的 Text 属性值设置为“1”。

（2）在页面后置文件 Default.aspx.cs 中编写单击按钮的事件代码如下。

```
protected void btnAdd_Click(object sender, EventArgs e)
{
    lbResult.Text = (int.Parse(lbResult.Text) + 1).ToString();
}
```

这样就实现了每次用户单击 Button 时，Label 显示的内容自动加 1。你一定会有一个疑问，前面讲过 HTTP 是无状态的，在回传过程中，Label 标签又是如何保存当前值的呢？在运行本任务的页面 Default.aspx 后，右击页面，在弹出的快捷菜单中选择“查看源文件”命令，会发现页面中被自动添加了两个<input>标签，代码如下。

```
< input type = "hidden" name = "__VIEWSTATE" id = "__VIEWSTATE"
      value = "/wEPDwUKMTI2NTY4ODI3MQ9kFgICAw9kFgICAQ8PFgIeBFRleHQFATVkZGTOAnyq
              EGo4jnQmgYtNfbrdDqxDvQ == " />
< input type = "hidden" name = "__EVENTVALIDATION" id = "__EVENTVALIDATION"
            value = "/wEWAgKOw4irDwKMk4HYD9BQEzLpvyyeBafwI8A + WxrDTLXh" />
```

其实，ASP.NET 使用__VIEWSTATE 来保存 Web 控件回传时状态的值。当 Web 窗体设置为 runat＝“server”时候，此窗体就被附加了一个隐藏的控件__VIEWSTATE，__VIEWSTATE 中存放了所有控件的状态值。当请求某页面时，ASP.NET 把所有控件的状态序列化成一个字符串，然后作为窗体的隐藏属性送到客户端，当客户端将页面回传时，ASP.NET 分析回传的窗体属性，并赋给控件对应的值。全过程由 ASP.NET 自动完成，开

发人员不需要处理。如果不需要维护视图状态，可以在页面的@Page指令中添加EnableViewState="false"或者将EnableViewState="false"属性添加至窗体中的控件中。

视图状态是用于与窗体元素对应的ASP.NET控件，同时，所有标准的ASP.NET控件也都可以维护自身状态。value属性中这个冗长的隐式字段含有窗体控件中所有以前的输入，Web窗体页面中包含所有控件的状态值都保留在一个ViewState控件的value属性中，只有将窗体回传给自身后，才能保留其视图状态。如果用户离开此页面，在访问一些其他网站后返回此页或者窗体像另一个页面发送，视图状态就会丢失。

任务4-2-3　实现简单登录操作

【任务描述】

使用Request对象和Response对象实现一个简单登录操作，用户登录页面如图4-13所示，实现用户名和密码检查，如果用户名和密码都输入正确，则将用户跳转到如图4-14所示的欢迎页面，在欢迎页面显示用户名、浏览器版本和浏览器语言等信息。



图4-13　用户登录页面

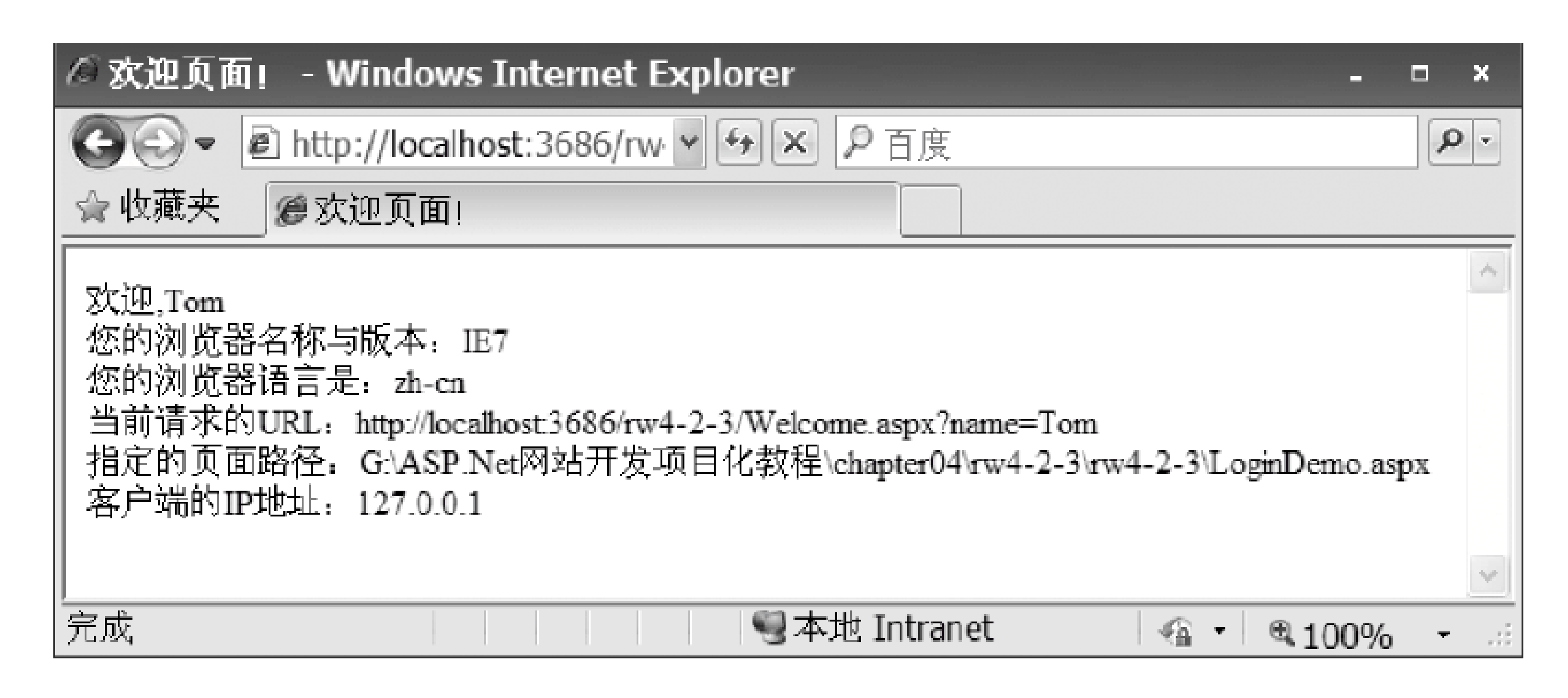


图4-14　登录成功欢迎页面

【任务实施】

(1) 创建网站项目rw4-2-3，通过右击网站项目，添加页面LoginDemo.aspx和Welcome.aspx。

(2) 在登录页后置代码文件LoginDemo.aspx.cs中编写单击"登录"按钮事件方法代码如下。

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (this.txtloginId.Text.Trim().Length == 0)
    {
        this.lblMessage.Text = "请输入用户名!";
        return;
    }
    if (this.txtLoginPwd.Text.Trim().Length == 0)
    {
        this.lblMessage.Text = "请输入密码!";
        return;
    }
    if (this.txtloginId.Text.Trim() == "Tom" && this.txtLoginPwd.Text.Trim() == "123")
    {
        Response.Redirect("Welcome.aspx?name=" + this.txtloginId.Text.Trim());
    }
    else
    {
        this.lblMessage.Text = "用户名/密码错误!";
    }
}
```

(3) 编写欢迎页面 Welcome.aspx 的 Page_Load 事件方法代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        string userName = Request.QueryString["name"];
        Response.Write("欢迎," + userName + "<br/>");
        Response.Write("您的浏览器名称与版本: ");
        Response.Write(Request.Browser.Type);
        Response.Write("<br>您的浏览器语言是: ");
        Response.Write(Request.ServerVariables["HTTP_ACCEPT_LANGUAGE"].ToString());
        Response.Write("<br>当前请求的 URL: ");
        Response.Write(Request.Url);
        Response.Write("<br>指定的页面路径: ");
        Response.Write(Server.MapPath("LoginDemo.aspx"));
        Response.Write("<br>客户端的 IP 地址: ");
        Response.Write(Request.ServerVariables["remote_addr"]);
    }
}
```

如果用户输入正确,LoginDemo.aspx 页面使用 Response.Redirect()将页面跳转到欢迎页面,并将用户姓名添加 URL 中,Welcome.aspx 页面用 Request.QueryString 从 URL 中获取用户名。本任务中的用户名和密码是固定的,实际项目中往往都是从数据库中读取,后续相关任务中会采用从数据库读取用户名和密码。

任务 4-2-4 实现系统级的登录功能

【任务描述】

在任务 4-2-3 的基础上结合 Session 和 Cookie 对象实现新知书店管理员登录功能。符合以下需求：

- 登录页面加载时，给出用户名的输入提示，如果客户端保存了用户名，显示用户名，如图 4-15 所示。



图 4-15 登录页面效果

- 实现用户名和密码的非空验证，如果都不为空进行用户名和密码的数据验证(为简化操作，本任务的用户名和密码仍然固定)，否则给出“请输入用户名和密码”的提示信息。
- 如果用户名和密码输入正确则跳转到新知书店的后台页面，并提示“欢迎，****”，否则给出“用户名或密码错误”的提示信息，如图 4-16 所示。



图 4-16 新知书店后台登录成功效果

【任务实施】

1. 思路分析

（1）使用 Cookie 判断客户端是否保存了用户名，如果 Cookie 为空则提示输入用户名，否则使用 Request. Cookie[Cookie 的名称]. Value 读取用户名并显示。

（2）使用 Session 保存用户名和密码。

（3）在加载后台首页时判断 Session 是否为空，如果为空使用 Response. Redirect()将用户重定向到登录页面。

2. Web 页面设计及编码

（1）创建网站项目 rw4-2-4，通过右击网站项目，新建文件夹 Images 并将登录页面图片素材复制到该目录下，添加页面 AdminLogin. aspx，在页面<head></head>标签对中编写样式代码如下。

```
<head>
  <style type="text/css">
    .login
    {
        position: absolute;
        top: 50%;
        left: 50%;
        margin: -250px 0 0 -250px;
    }
    .login_t
    {
        margin: 0 auto;
        width: 598px;
        height: 78px;
        background: url(images/login_03.gif) no-repeat;
    }
    .login_m
    {
        margin: 0 auto;
        width: 598px;
        height: 142px;
        background: url(images/login_05.gif) no-repeat;
    }
    .login_b
    {
        margin: 0 auto;
        width: 598px;
        height: 150px;
        padding-top: 24px;
        background: url(images/login_06.gif) no-repeat;
        text-align: center;
    }
```

```
        .login_b p
        {
            margin: 12px 0;
        }
        .login_input
        {
            width: 160px;
            height: 20px;
            margin - left: 6px;
            line - height: 20px;
            border: 1px solid #999;
        }
        .login_sub
        {
            width: 67px;
            height: 19px;
            background: url(images/login_sub.gif) no - repeat;
            text - align: center;
            border: none;
            line - height: 22px;
        }
    </style>
< head >
```

（2）从工具箱拖入两个 TextBox 控件和两个 Button 控件至页面 AdminLogin.aspx，并添加代码如下。

```
< form id = "Login" name = "Login" method = "post" runat = "server">
  < table  align = "center" >
    < tr >
        < td >用户名</td >
        < td >< asp:TextBox runat = "server" ID = "txtLoginId" CssClass = "login_input"></asp:
TextBox ></td >
    </tr >
    < tr >
        < td >密   码</td >
        < td >< asp:TextBox runat = "server" ID = "txtLoginPwd" TextMode = "Password"
                  CssClass = "login_input"></asp:TextBox ></td >
    </tr >
    < tr >
        < td colspan = "2">
            < asp:Button runat = "server" ID = "btnConfirm" OnClick = "btnConfirm_Click"
                      Text = "提交" CssClass = "login_sub" />
            < asp:Button runat = "server" ID = "btnCancel" OnClick = "btnCancel_Click"
                      Text = "重置" CssClass = "login_sub" />
        </td >
    </tr >
  </table >
</form >
```

(3) 在登录页后置代码文件 AdminLogin.aspx.cs 中编写单击"提交"、"重置"按钮的事件方法及页面 Page_Load 事件方法代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        if (Request.Cookie["UserName"] == null)
        {
            this.txtLoginId.Text = "请在此输入用户名";
        }
        else
        {
            this.txtLoginId.Text = Request.Cookie["UserName"].Value;
        }
        this.txtLoginPwd.Text = "请在此输入密码";
    }
}
protected void btnConfirm_Click(object sender, EventArgs e)
{
    string strMsg = string.Empty;
    if (this.txtLoginId.Text.Trim() == "admin" && this.txtLoginPwd.Text.Trim() == "123456")
    {
        Response.Cookie["UserName"].Value = this.txtLoginId.Text.Trim();
        Response.Cookie["UserName"].Expires = DateTime.Now.AddDays(10);
        //以下注释的三行代码为实现 Cookie 的另外一种方式
        //HttpCookie hcCookie = new HttpCookie("UserName", this.txtLoginId.Text.Trim());
        //hcCookie.Expires = DateTime.Now.AddDays(1);
        //Response.Cookie.Add(hcCookie);
        UserInfo userInfo = new UserInfo();
        userInfo.UserName = this.txtLoginId.Text.Trim();
        userInfo.UserPwd = this.txtLoginPwd.Text.Trim();
        Session["user"] = userInfo;
        Response.Redirect("Admin/Default.aspx?name=" + this.txtLoginId.Text.Trim());
    }
    else
    {
        Response.Write(strMsg);
    }
}
protected void btnCancel_Click(object sender, EventArgs e)
{
    this.txtLoginId.Text = String.Empty;
    this.txtLoginPwd.Text = String.Empty;
}
```

(4) 通过右击网站项目 rw4-2-4,新建文件夹 Admin,并在文件夹 Admin 下新建两个子文件夹 Css、Images,用于存放后台首页的样式文件 admin.css 和图片资源;通过右击 Admin 文件夹,添加新知书店后台首页 Default.aspx,拖入一个 Label 标签,修改 ID 为

blCome，并编写代码如下。

```
<head runat = "server">
    <title>新知书店 - 管理后台</title>
    <link href = "CSS/admin.css" rel = "stylesheet" type = "text/css" /><% -- 从 Css 文件夹导
入样式文件 -- %>
</head>
<body>
    <form id = "Form1" runat = "server">
    <div id = "header">
        <img src = "images/admin_top.gif" alt = "" /></div>
    <div id = "main">
        <div id = "opt_list">
            <h1>管理员，您好!</h1>
        </div>
        <div id = "opt_area"><br />
            <asp:Label ID = "lblCome" runat = "server" Text = "Label"></asp:Label>
        </div>
    </div>
    </form>
</body>
```

（5）编写后台首页页面后置代码文件 Default.aspx.cs 中的代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["user"] == null)
    {
        Response.Redirect("~/AdminLogin.aspx");
    }
    if (!IsPostBack)
    {
        UserInfo user = Session["user"] as UserInfo; //使用 Session 实现
        lblCome.Text = "欢迎," + user.UserName;
    }
}
```

任务 4-2-5　统计网站同时在线人数及页面点击次数

【任务描述】

在任务 4-2-4 的基础上，统计新知书店后台同时在线人数及后台首页浏览次数(刷新页面也是一次新的浏览)。

【任务实施】

（1）创建网站项目 rw4-2-5，将任务 4-2-4 下的页面及文件夹全部复制到网站项目 rw4-2-5，拖放两个 Label 标签至 Admin 文件夹下的 Default.aspx 页面中，并在网站根目录下添加 Global.asax 文件。

（2）在 Application_Start 事件中使用 Application["UserOnLineCnt"] = 0 初始化同

时在线人数,Application["PageClickCnt"] = 0 初始化网页浏览次数,代码如下。

```
void Application_Start(object sender, EventArgs e)
{
    //在应用程序启动时运行的代码
    Application.Lock();
    Application["UserOnLineCnt"] = 0;       //记录同时在线人数
    Application["PageClickCnt"] = 0;        //记录页面点击次数
    Application.UnLock();
}
```

(3) 在 Session_Start 事件中增加在线人数,在 Session_End 事件中减少在线人数,减少人数时需要判断当前 Application["UserOnLineCnt"]是否大于零,代码如下。

```
void Session_Start(object sender, EventArgs e)
{
    //在新会话启动时运行的代码
    Application.Lock();
    Application["UserOnLineCnt"] = (int)Application["UserOnLineCnt"] + 1;
    Application.UnLock();
}

void Session_End(object sender, EventArgs e)
{
    //在会话结束时运行的代码
    //注意: 只有在 Web.config 文件中的 sessionstate 模式设置为
    //InProc 时,才会引发 Session_End 事件。如果会话模式
    //设置为 StateServer 或 SQLServer,则不会引发该事件
    if (((int)Application["UserOnLineCnt"]) > 0)
    {
        Application.Lock();
        Application["UserOnLineCnt"] = (int)Application["UserOnLineCnt"] - 1;
        Application.UnLock();
    }
}
```

(4) 在页面的后置代码文件 Default.aspx.cs 中编写实现统计网页访问次数的代码,这里回传不能算是一次浏览,实现代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["user"] == null)
    {
        Response.Redirect("~/AdminLogin.aspx");
    }

    if (!IsPostBack)
    {
        UserInfo user = Session["user"] as UserInfo;   //使用 Session 实现
```

```
            lblCome.Text = "欢迎," + user.UserName;
            //显示网站在线人数,统计由 Session_Start 事件实现,即建立会话增加 1 人
            if (Application["UserOnLineCnt"]!= null)
            {
                lbUserOnLineCnt.Text = "网站在线人数: " + (int)Application["UserOnLineCnt"];
            }
            //实现浏览网页次数统计
            Application.Lock();
            Application["PageClickCnt"] = (int)Application["PageClickCnt"] + 1;
            Application.UnLock();
            lbPageClickCnt.Text = "本网页浏览(刷新)次数:" + (int)Application["PageClickCnt"];
        }
    }
```

(5) 运行 AdminLogin.aspx 页面,输入正确的用户名和密码,登录至新知书店后台首页 Default.aspx,刷新一次页面,效果如图 4-17 所示,更换用户再次登录并刷新页面几次后,效果如图 4-18 所示。



图 4-17 统计在线人数及网页浏览次数(admin)



图 4-18 统计在线人数及网页浏览次数(Tom)

4.3 项目实训

实现"博客系统"的登录模拟功能(不基于数据库)

【需求说明】

- 参考任务 4-2-4 为"博客系统"创建用户登录页面 login.aspx,用 Session 来保存用户登录信息,然后跳转到欢迎页面,再在欢迎页面读取 Session 的值,并进行验证,要求只有用户名为 MyBlog、密码为 123456 的用户才能登录,登录成功后跳转到欢迎页面 Welcome.aspx,并用 Session 输出用户信息。(提示:创建一个 UsersInfo 类,用来保存用户信息;利用 Session 来保存 userInfo 对象的信息;在欢迎页面读取 Session 的值,并对其进行验证,验证失败返回登录页。)
- 为了跟踪注册会员访问"博客系统"的情况,要求把成员的访问记录保存在用户的浏

览器端。并能记录访问者的最近访问次数，期限为一个月，并新建一个成员管理页面 UserMsg. aspx 查看成员的访问记录。

4.4　单元小结

本单元讲解了页面生命周期、页面传值及 ASP.NET 内置对象，包括如何创建 ASP.NET 内置对象和使用 ASP.NET 内置对象。本单元重点介绍了以下 6 个对象。

- Request 对象：用来获取客户端信息。
- Response 对象：可以向客户端输出。
- Session 对象：记载特定客户的信息。
- Application 对象：存储 ASP.NET 应用程序中多个会话和请求之间的全局共享信息。
- Server 对象：专为处理服务器上的特定任务。
- Cookie 对象：一种可以在客户端保存信息的方法。

Web 应用程序从本质上来讲是无状态的，为了维护客户端的状态，还重点阐述了如何使用 Session、Cookie、Application 等 ASP.NET 内置对象进行客户端状态的维护。本单元知识体系如图 4-19 所示。



图 4-19　系统对象与数据传递知识体系

4.5　单元练习题

一、选择题

1. 下面（　　）文件主要定义应用开始和结束、会话开始和结束、请求开始和结束等事件发生时，要做的事情。

A. web. config　　B. Global. inc　　C. Config. asax　　D. Global. asax

2. 一个 ASP.NET 应用程序中一般只有(　　)个 Global.asax 文件有效。

A. 0　　B. 1　　C. 若干　　D. 以上都不对

3. 在一个 ASP.NET 应用程序,希望在每一次新的会话开始时,进行一些初始化任务。应该在(　　)事件中编写代码。

A. Application_Start　　B. Application_BeginRequest

C. Session_Start　　D. Session_End

4. 下列选项中,只有(　　)不是 Page 指令的属性。

A. CodePage　　B. Debug　　C. namespace　　D. Language

5. 在一个名为 Login 的 Web 网页中,先需要在其 Page_Load 事件中判断该页面是否回传,需要使用下列(　　)属性。

A. Page.IsCallback　　B. Page.IsAsync

C. Page.IsPostBack　　D. Login.IsPostBack

6. (　　)事件在页面被加载的时候,自动调用该事件。

A. Page_Load　　B. Page_UnLoad

C. Page_OnLoad　　D. Page_Submit

7. 下面程序段执行完毕后,页面显示的内容是(　　)。

```
Response.Write("Hello");
Response.End();
Response.Write("World");
```

A. HelloWorld　　B. World　　C. Hello　　D. 出错

8. 下面的(　　)方法用于将客户浏览器重新定向到一个新的 URL 地址。

A. Redirect　　B. BinaryRead　　C. UrlPathEncode　　D. UrlDecode

9. 使用(　　)对象的 SaveAs 方法可以将 HTTP 请求保存到磁盘上。

A. Request　　B. Response　　C. Session　　D. Application

10. 一家在线测试中心 TestKing 公司创建一个 ASP.NET 应用程序。在用户结束测试后,这个应用程序需要在用户不知道的情况下,提交答案给 ProcessTestAnswers.aspx 页。这 ProcessTestAnswers.aspx 页面处理这答案,但不提供任何显示消息给用户。当处理完成时,PassFailStatus.aspx 页面显示结果给用户。在 PassFailStatus.aspx 页面中加(　　)代码,来执行 ProcessTestAnswers.aspx 页面中的功能。

A. Server.Execute("ProcessTestAnswers.aspx")

B. Response.Redirect("ProcessTestAnswers.aspx")

C. Response.WriteFile("ProcessTestAnswers.aspx")

D. Server.Transfer("ProcessTestAnswers.aspx",True)

11. 一个应用程序中一般有(　　)个 web.config 文件有效。

A. 0　　B. 1　　C. 若干　　D. 以上都不对

12. 在名为 Login 的页面的 Page_Error 事件中捕获了一个未处理的异常,现需要清除刚产生的异常,需要使用下列(　　)语句。

A. HttpServerUtiliity.ClearError()　　B. Page.ClearError()

C. Login.ClearError()　　D. Server.ClearError()

13. 在一个 ASP.NET 的网站中,如果需要在应用程序级捕获未处理的异常,应该使用下列(　　)事件。

A. Response_Error　　B. Server_Error

C. Application_Error　　D. Page_Error

14. Request 对象中获取 Get 方式提交的数据的方法是(　　)。

A. Cookie　　B. ServerVariables

C. QuerySttring　　D. Form

15. 创建一个显示金融信息的 Web 用户控件。如果希望该 Web 用户控件中的信息能在网页的请求之间一直被保持,应该采取(　　)方法。

A. 设置该 Web 用户控件的 PersistState 属性为真

B. 设置该 Web 用户控件的 EnableViewState 属性为真

C. 设置该 Web 用户控件的 PersistState 属性为假

D. 设置该 Web 用户控件的 EnableViewState 属性为假

16. Session 对象的默认有效期为(　　)分钟。

A. 10　　B. 15　　C. 20　　D. 30

17. 下面程序段执行完毕,页面显示的内容是(　　)。

```
string strName;
strName = "user_name";
Session["strName"] = "Mary";
Session[strName] = "John";
Response.Write(Session["user_name"]);
```

A. Mary　　B. John

C. user_name　　D. 语法有错,无法正常运行

18. 下列(　　)对象经常用来制作网页计数器。

A. Response　　B. Application　　C. Request　　D. Session

19. 在同一个应用程序的页面 1 中执行 Session.Timeout=30,那么在页面 2 中执行 Response.Write(Session.Timeout),则输出值为(　　)。

A. 15　　B. 20　　C. 30　　D. 25

20. Application 对象的默认有效期为(　　)。

A. 10 天　　B. 15 天

C. 20 天　　D. 从网站启动到终止

21. 下面代码实现一个站点访问量计数器,空白处的代码为(　　)。

```
void ________(object sender, EventArgs e)
{
Application.Lock();
Application["AccessCount"] = (int)Application["AccessCount"] + 1;
Application.UnLock();
}
```

A. Application_Start　　　　B. Application_Error

C. Session_Start　　　　D. Session_End

二、填空题

1. 列举 ASP.NET 中的七个内置对象：________、________、________、________、________、________、________。

2. 应用程序开始时，调用________事件；应用程序结束时，调用________。

3. 一次新的会话开始时，调用________事件；会话结束时，调用________事件。

4. ________方法可获得网站根目录的物理路径。

三、问答题

1. 简述 Global.asax 文件的结构。Web 应用程序可以在哪些目录中放置此文件？

2. ASP.NET 页面包含哪些内置对象？

3. 简述 ASP.NET 网页文件由哪儿部分组成。

4. 试说明什么是 Application 和 Session 对象，其差异是什么？如果存储用户专用信息，应该使用哪个对象变量来存储？

5. 什么是 Cookie？如何创建和读取 Cookie 对象？

6. Application 对象的 Lock 方法和 UnLock 方法具有什么作用？

单元5 搭建风格统一的Web站点

教学目标：

- 能创建母版页及基于母版页的内容页。
- 了解母版页的优点以及与普通页面的区别。
- 学会创建网站地图文件。
- 掌握使用 SiteMapPath 控件实现“面包屑导航”功能。
- 掌握使用 TreeView 控件的节点编辑器设置树状导航的方法。
- 掌握使用 TreeView 控件和递归法动态实现树状导航。

5.1 知识准备

5.1.1 CSS 样式控制

CSS(Cascading Style Sheets,级联样式表)是用于控制网页样式并允许将样式信息与网页内容分离的一种标记性语言,是非常重要的页面布局方法,也是最高效的页面布局方法。

1. 页面中使用 CSS 的三种方法

CSS 被设计用来与 HTML 联合建立网页,它不能独立运行,需要依附到页面上才能发挥作用。通常在网页中 CSS 规定了如下 3 种定义样式的方法。

- 内联式：直接将样式控制放置在单个 HTML 元素内。
- 嵌入式：在网页的 head 部分定义样式。
- 外部链接式：以扩展名.css 的文件保存被称为 CSS 文件的样式定义,将 CSS 文件链接到相应的网页中。

1）内联式样式

内联样式直接将 CSS 放在某个 HTML 标签中,通过使用 style 属性设置,一般形式为：

```
style="属性名1:值1;属性名2:值2;…"
```

例如：

```
<body>
      <div style="color: Red; background-color: blue">内联式样式示例</div>
</body>
```

属性名与属性值之间用“:”分隔，如果一个样式中有多个属性，各属性之间用分号“;”隔开。用内联式的方法进行样式控制固然简单，但是其维护过程却是非常复杂和难以控制的。

【示例 5-1】 style 属性的内联式样式演示。

(1) 在 ch05 网站项目中创建文件 StyleInCss.aspx

(2) 在页面文件 StyleInCss.aspx 中添加如下代码：

```
<body>
    <div style="font-size: 16px;">这是一段测试文字 1</div>
    <div style="font-size: 16px; font-weight: bolder">这是一段测试文字 2</div>
    <div style="font-size: 16px; font-style: italic">这是一段测试文字 3</div>
    <div style="font-size: 20px; font-variant: small-caps">This is My First CSS code</div>
    <div style="font-size: 14px; color: red">这是一段测试文字 5</div>
</body>
```

2) 嵌入式样式

在网页的 head 部分直接实现 CSS 样式，即在<head>与</head>标签内定义样式，以<style>开始，</style>结束。CSS 规则由两部分组成：选择符和声明。声明由属性名和属性值组成。简单的 CSS 规则如下：

```
选择符{属性名 1: 值 1; 属性名 2: 值 2; …}
例如: p { color : Green; }
```

p(段落标签)为选择符，color(颜色)是 p 的属性名，green(绿色)是 color 的属性值。该规则声明所有段落标签的内容应该将 color 属性设置为绿色，即所有<p>中文本将变成绿色。

【示例 5-2】 style 属性的嵌入式样式演示。

(1) 在 ch05 网站项目中创建文件 StyleInLinkCss.aspx。

(2) 在页面文件 StyleInLinkCss.aspx 中添加如下代码：

```
<head id="Head1" runat="server">
    <title>示例 5-2</title>
    <style type="text/css">
        body
        {
            text-align: center;
            font-family: @微软雅黑;
            font-size: xx-large;
        }
        div
        {
            color: Red;
            background-color: blue;
```

```
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
    <div>嵌入式样式示例</div>
    </form>
</body>
```

3）链接式样式

在页面中使用 CSS 最常用的方法是链接式样式。利用这种方法可以在网页中调用已经定义好的样式表文件(.css 文件)。

与嵌入式相比，链接式可以将定义好的样式在网站的多个页面上重复使用，提高了开发效率，降低了维护成本，同时也实现了将页面结构和表现彻底分离，最适合大型网站的外观设计。本书贯穿项目“新知书店”网站的页面样式控制采取链接式样式。

2. 样式规则

无论是定义内嵌式样式还是链接式样式，每个样式的定义格式相同：

```
选择符{属性名 1:值 1;属性名 2:值 2; …}
```

其中，选择符是指样式定义的对象，可以是 HTML 标记元素、用户自定义的类、用户自定义的 id、伪类、具有层次关系的样式规则及并列的样式选择符等。

1）元素选择符

任何 HTML 元素都可以是一个 CSS 的元素选择符，例如，div{color ：red}，该样式规则中的元素选择符是 div，div 块内的所有文字颜色为红色。

2）类选择符

类选择符用于定义页面上的相关 HTML 元素组，使它们具有合适的相同样式规则。创建类时，用户需要给它命名，命名时最好使用字母和数字。

定义了类之后，用户可以使用它作为 CSS 的选择符。类选择符以“.”为起始标记，一般格式为：

```
.类选择符{属性名 1:值 1;属性名 2:值 2; …}
```

例如：

```
.c1 { color : Red; }
.c2 { font-size : large; }
```

上面定义了两个类，类 c1 定义了颜色属性，类 c2 定义了字体大小属性。

在 HTML 文档中可以按下列方式引用：

```
<div>
    <h1 class="c1">通知</h1>
    <p class="c2">将与今天下午 2 点召开各部门会议.</p>
</div>
```

标签<h1>中的文本颜色为红色，标签<p>中的字体大小为“large”。因为它们各自的 class 属性值为类 c1 和类 c2。

3）id 选择符

只有在页面上的标签才能具有给定的 id，它必须是唯一的，并只用于指示该元素。

下面的例子中标签<a>定义了一个 id 属性，值是"next"。

```
<a href="next.htm" id="next">下一步</a>
```

在 CSS 中，id 选择符由 id 值前面的“#”(井号)符号指示，例如：

```
#next { font-size : large; }
```

在实际应用中，用户应如何选取类选择或 id 选择符设置样式呢？

类选择符更灵活，id 选择符能完成的它都能完成，甚至比 id 选择符能完成的还要多。如果想重用样式，用户也可以使用类选择符来完成。但是用 id 选择符就完成不了，因为 id 值在页面文档中必须是唯一的，即只有一个元素具有该值。

注意：如果在一个元素的样式定义中，既引用了元素选择符，又引用了类选择符和 id 选择符，则 id 选择符的优先级最高，其次是类选择符，元素选择符的优先级最低。

4）伪类

伪类可以看作是一种特殊的类选择符，是能被支持 CSS 的浏览器自动所识别的特殊选择符。它的最大的用处就是可以对链接在不同状态下定义不同的样式效果。

在 CSS 中用 4 个伪类来定义链接样式，分别是 a:link、a:visited、a:hover 和 a:active，例如：

```
a:link {color: #FF0000}              /* 未被访问的链接 红色 */
a:visited {color: #00FF00}           /* 已被访问过的链接 绿色 */
a:hover {color: #FFCC00}             /* 鼠标悬浮在上的链接 橙色 */
a:active {color: #0000FF}            /* 鼠标点中激活链接 蓝色 */
```

以上语句分别定义了链接、已访问过的链接、鼠标停在上方时、单击鼠标时的样式。注意，必须按以上顺序书写，否则不能按预期效果显示。

5）包含选择符

包含选择符用于定义具有层次关系的样式规则，它由多个样式选择符组成，选择符之间用空格隔开。一般格式为：

```
选择符 1 选择符 2 … {属性名 1:值 1;属性名 2:值 2; …}
```

例如，div h1{ color : red }，这种方式只对 div 中包含的 h1 起作用，对单独的 div 或 h1

均无效。

6）并列选择符

如果有多个不同的样式选择符的样式相同，则可以使用并列选择符简化定义，每个样式选择符之间用逗号隔开。一般格式为：

```
选择符 1,选择符 2, …{属性名 1:值 1;属性名 2:值 2; …}
```

例如：

```
.classone, #bb, h1{color : red}
```

5.1.2　页面框架

1. “新知书店”项目概况

“新知书店”是一个电子商务网站，它以图书的在线销售为主要内容，根据单元 1 中的分析，“新知书店”功能模块可以分为前台页面、管理端后台页面和用户后台页面三大部分。管理端后台有页面时在根目录下的 Admin 文件夹下；用户后台所有页面在 Membership 文件夹中；前台页面直接放在站点根目录下。一般而言，每一部分都有各自的布局风格，接下来将使用所学知识和技能完成“新知书店”的项目开发任务。

2. 网页布局和框架技术

1）网页布局

一个积累了大量用户的网站，仅靠网站的内容来吸引用户眼球是远远不够的，网站的风格和良好的用户体验也起着至关重要的作用。网站就像网上的家，也需要装修、设计，这就需要合理布局，典型的页面布局有栏式结构和区域结构。

栏式结构是很常见的页面结构，其特点是简单实用、条理分明、格局清晰严谨，适合信息量大的页面。图 5-1 列出了几种常见的栏式结构。

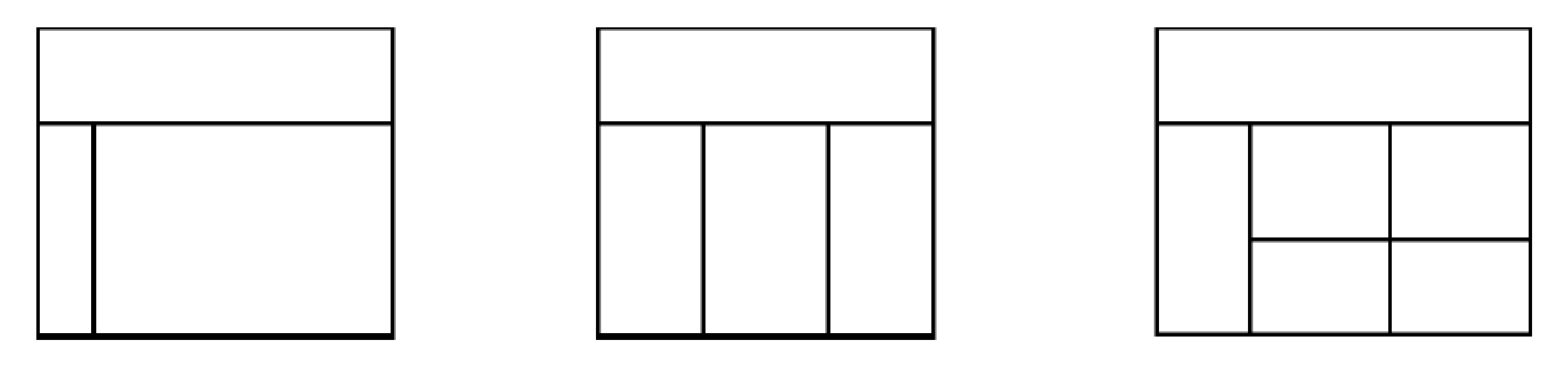


图 5-1　常见的网页栏式结构

区域结构现在使用比较少，其特点是页面精美、主题突出、空间感很强，不过仅适合信息量比较少的页面。图 5-2 是一个区域结构的样例，页面被划分成了多个区域，有很大的空间留给背景。

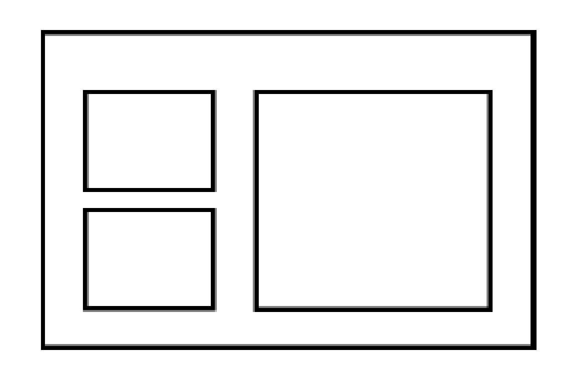


图 5-2　区域结构示例

实现页面布局，现在一般使用 DIV＋CSS。早期曾大量使用<table>作为布局方式，由于 DIV＋CSS 方式具有代码简洁等优势，<table>布局方式逐渐被弃用。

注意：<table>用于页面布局越来越少，但并不代表<tabel>标签应被丢弃，在 DIV 布局的页面上，常常会有<table>的表格。在许多项目中，往往用 DIV 实现页面的整体布局，而页面中列表的显示还是常常使用<table>标签。

2）Frameset 和 Iframe

前面介绍过框架集(<frameset>)和内嵌框架<iframe>。Frameset 是指页面各个窗口全部用框架实现，使用 frameset 可以在同一个浏览器中显示多个页面的集合。Iframe 是指页面中的部分内容用框架实现。

使用页面框架的优势如下：

- 框架能将多个页面组合显示，并且各个页面之间相互独立。
- 页面的复用实现了站点风格的统一。
- 页面结构清晰便于用户的使用。

在 ASP.NET 中，也可以使用框架技术完成页面设计。如图 5-3 所示是使用内嵌框架<iframe>实现的“新知书店”管理后台首页，左侧方框引用的是管理后台菜单页面 adminMenu.aspx，右侧方框引用的是管理后台内容页面 adminContent.aspx。



图 5-3　使用框架构建的“新知书店”管理后台首页

管理后台首页中的关键代码如下：

```
<form id="form1" runat="server">
    <div>
        <img src="images/admin_top.gif" alt="" />
    </div>
    <div>
        <iframe name="admin_menu" src="adminMenu.aspx" frameborder="0"
```

```
            scrolling = "no"   width = "210"   height = "590"></iframe>
        < iframe name = "admin_content" src = "adminContent.aspx" frameborder = "0"
            scrolling = "no"   width = "760" height = "590"></iframe>
    </div>
</form>
```

虽然页面框架技术有诸多优势,但在实际开发中页面框架技术也存在如下问题:

- Iframe 标签的内容不利于搜索引擎的收录。
- 支持页面框架技术的浏览器不多。
- Iframe 中链接的页面会增加请求。
- 页面设计不直观,不便于页面编写。

ASP.NET 中的母版技术可以很好地解决上述问题。

5.1.3　母版页

在 Web 应用开发过程中,经常会遇到 Web 应用程序中的很多页面的布局都相同这种情况。在 ASP.NET 中,可以使用 CSS 和主题减少多页面的布局问题,但是 CSS 和主题在很多情况下还无法胜任多页面的开发,这时就需要使用母版页。

1. 母版页和内容页的概念

母版页是用于设置页面外观的模板,是一种特殊的 ASP.NET 网页文件,同样也具有其他 ASP.NET 文件的功能,如添加控件、设置样式等,只不过其扩展名是.master。在母版页中,界面被分为公用区和可编辑区,公用区的设计方法与一般页面的设计方法相同,可编辑区用 ContentPlaceHolder 控件预留出来。

引用母版页的.aspx 页面成为内容页,在内容页中,母版的 ContentPlaceHolder 控件预留的可编辑区会被自动替换为 Content 控件,开发人员只需要在 Content 控件区域中填充内容即可,在母版页中定义的其他标记将自动出现在引用该母版页的.aspx 页面中,母版页的部分以灰色显示,表示不能修改这些内容。

母版页和内容页的关系如图 5-4 所示。

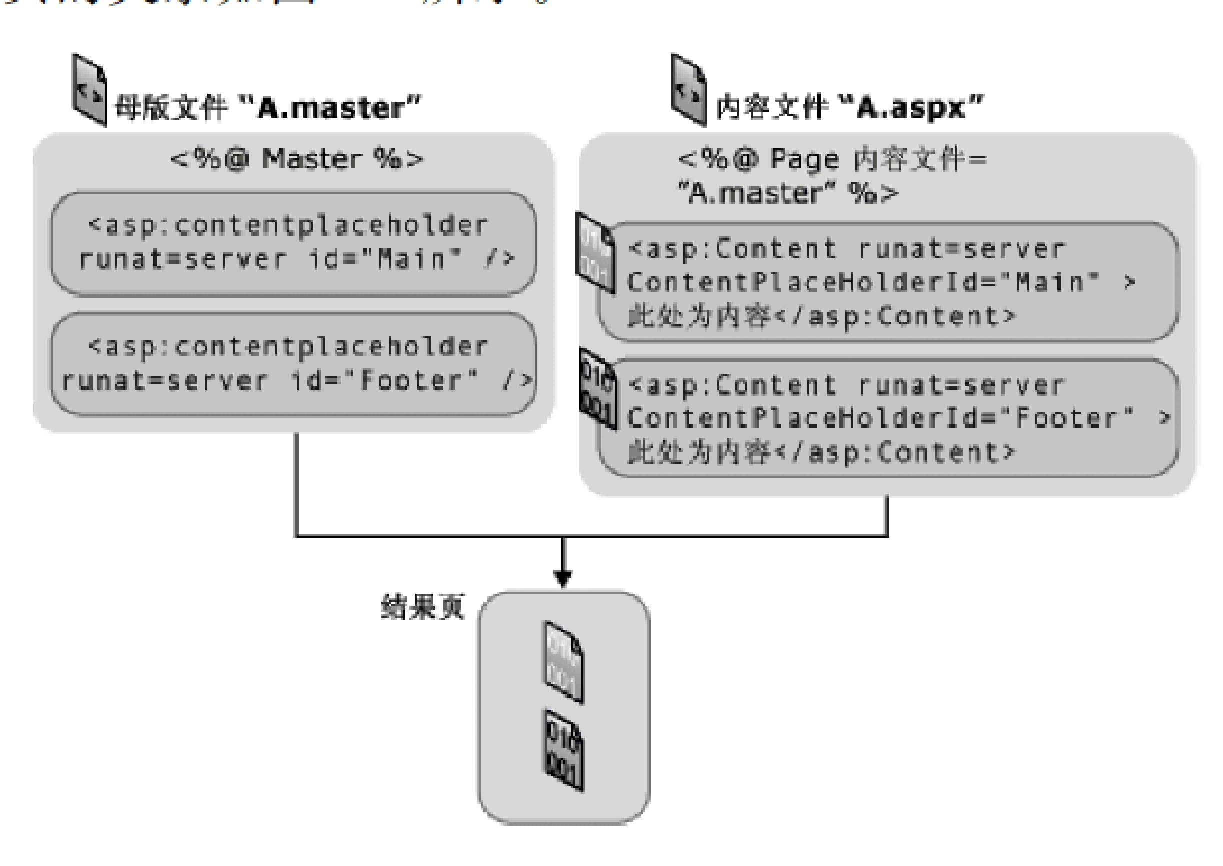


图 5-4　母版页和内容页的关系

从图 5-4 中可以总结出母版页的工作过程：

(1) 用户在浏览器中通过内容页的 URL 来请求访问 Web 页面。

(2) 获取该页后，读取页面的 Page 指令。如果该指令引用一个母版页，则读取相应的母版页。如果是第一次请求这两个页，则两个页都要进行编译。

(3) 将内容页中各个 Content 控件的内容合并到母版页中相应的 ContentPlaceHolder 控件中，生成结果页。

(4) 用户浏览器中呈现服务器返回的由母版页与内容页合并的结果页。

步骤(2)～(4)对用户来说是透明的，由服务器自动完成，用户只需提供内容页的 URL 即可。

2. 母版页的特点

使用母版页，可以为 ASP.NET 应用程序页面创建一个通用的外观。开发人员可以利用母版页功能创建一个布局，然后在多个页面中应用该布局。母版页具有以下特点。

- 有利于站点修改和维护，降低开发人员的工作强度。由于公共内容被集中于母版页中，因此，只要修改母版页，就可以快速完成站点修改和维护任务，这在很大程度上提高了工作效率。
- 提供高效的内容整合能力。在母版页中允许添加文字、控件等 Web 元素，同时，也可以为这些 Web 元素添加相应的事件处理程序。例如，只需要在母版页中添加一个服务器控件事件处理程序，站点内所有引用该母版页的网页都将获得同样的应用效果。
- 有利于实现页面布局。内容页对应于母版页中的一处位置，这在很大程度上有利于页面的布局工作。
- 提供一种便于利用的对象模型。由内容页和母版页组成的对象模型，能够为应用程序提供一种高效、易用的实现方式，并且这种对象模型的执行效率较以往的处理方式有了很大的提高。

3. 母版页的创建及使用

创建母版页的方法和创建一般页面的方法非常相似，区别在于母版页无法单独在浏览器中查看，而必须通过创建内容页才能浏览，示例 5-3 演示了母版页的创建及使用过程。

【示例 5-3】 母版页的创建及使用演示。

(1) 在 ch05 网站项目中创建文件夹 ch5_3，右击该文件夹，从弹出的快捷菜单中选择“添加新项”命令，在出现的对话框中选择“母版页”项目，向项目中添加一个母版页，如图 5-5 所示。可以看到，母版页创建也有“将代码放在单独的文件中”复选框。如果需要在母版页中编写后台代码，就可以选中该选项。

新创建的母版页上面默认有两个 ContentPlaceHolder 控件，分别是预留给内容页的头部和主体部分显示的控件，可以把每个 ContentPlaceHolder 控件理解成一个“内容占位符”，至于这个占位符中到底会显示哪些内容，则由后续介绍的内容页中的 Content 控件决定。



图 5-5　添加母版页

创建后的母版页代码如下所示：

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
    <title>新知书店</title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div>
    新知书店
    <asp:ContentPlaceHolder id="cphContent" runat="server">
    </asp:ContentPlaceHolder>
    </div>
    联系我们
    </form>
</body>
</html>
```

头部的 ContentPlaceHolder 控件是 ASP.NET 3.5 新增的内容，可以用它为不同的内容页添加相应的样式表。

在母版页上添加控制显示的控件，如，分别在页头和页尾放置一个 Label 控件用于显示文本，如图 5-6 所示。

观察母版页的源代码，在页面的顶部有一个@Master 声明，而不是通常在 ASP.NET 页顶部看到的@Page 声明，指令如下：

```
<%@ Master Language="C#" AutoEventWireup="true" CodeFile="......"  Inherits="......" %>
```



图 5-6　母版页的样式

普通 ASP.NET 页面的声明如下：

```
<%@ Page Language="C#"  AutoEventWireup="true" CodeFile="......"  Inherits="......" %>
```

提示：在实际开发中，美工会制作静态页面，只需要将美工制作的页面代码粘贴到母版页即可，不过，注意至少保留一个 ContentPlaceHolder 控件，本示例中将内容部分的 ContentPlaceHolder 控件的 Id 设置为 cphContent。

(2) 在 ch05 网站项目中的文件夹 ch5_3 上右击，从弹出的快捷菜单中选择“添加新项”命令。在弹出的对话框中选择“Web 窗体”选项，在“名称”文本框中输入 Default.aspx，选中“选择母板”复选框，如图 5-7 所示，然后单击“添加”按钮，会创建刚才所创建母版页 Default.Master 的内容页 Default.aspx。

图 5-7　添加内容页

新添加的内容页在设计模式下可以看到母版页上的内容，而且除了 Content 控件之外其他内容是不可编辑的，如图 5-8 所示。查看内容页的源代码发现内容页的代码似乎比我们想象的要简单，没有标准的 HTML 格式，只有 Content 控件，代码如下所示。

```
<%@ Page Title="" Language="C#" MasterPageFile="~/ch5_3/Default.master"
    AutoEventWireup="true"  CodeFile="Default.aspx.cs" Inherits="ch5_3_Default"  %>
```

```
<asp:Content ID="Content1" ContentPlaceHolderID="head" runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cphContent" runat="Server">
</asp:Content>
```

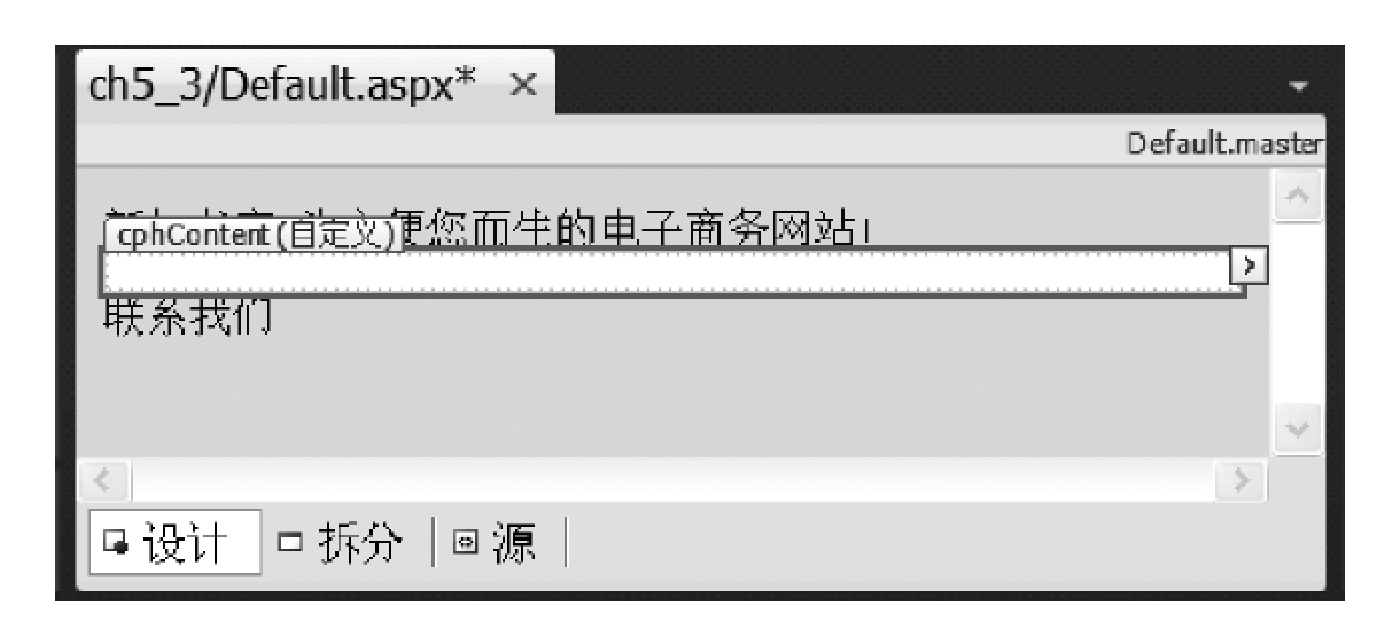


图 5-8 内容页的显示

- MasterPageFile：用于指定所使用母版页的路径。
- Title：用于设置内容显示的标题。
- ContentPlaceHolderID：用户控制该 Content 控件在页面中的位置，即指定所对应的母板页中 ContentPlaceHolder 控件的 ID，如果指定的 ID 在母版页中不存在，将会发生错误，如果存在母版页中的 ContentPlaceHolder 控件，该控件将被内容页中的 Content 控件所替代。

4. 获取母版页上控件的值

通过添加公共属性或使用 FindControl()方法可以实现在内容页中访问母版页中的属性。下面通过开发一个根据用户年龄段提供不同主题网站的示例演示如何在内容页中获取母版页上控件的值。

1）通过添加公共属性

首先在网站项目 ch05 中的文件夹 getMasterValue 下创建母版页 MainMaster.master，在母版页中添加一个 RadioButtonList 控件，母版页中的主要源代码如下：

```
<form id="form1" runat="server">
    <div>
        <asp:RadioButtonList ID="rblRole" runat="server" RepeatDirection="Horizontal"
            AutoPostBack="True">
            <asp:ListItem Value="0">学生族</asp:ListItem>
            <asp:ListItem Value="1">上班族</asp:ListItem>
        </asp:RadioButtonList>
        <hr/>
        <asp:ContentPlaceHolder id="cphContent" runat="server">
        </asp:ContentPlaceHolder>
    </div>
</form>
```

由于母版页本身也是个类，所以可以在其中添加公共属性，本例中在母版页 MainMaster.master.cs 中添加如下代码：

```
public string MasterValue
{
  get { return this.rblRole .SelectedValue; }
}
```

下面需要在内容页中编写访问代码，首先基于母版页创建 property.aspx 内容页，然后在内容页 property.aspx 的隐藏 cs 文件中通过调用 Master 属性访问母版页中的公共成员，代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    string strValue = this.Master.MasterValue.ToString ();
    if (strValue.Equals ("0"))
    {
        this.lblText.Text  =  "高校 BBS 求职信息";
    }
    else if (strValue.Equals("1"))
    {
        this.lblText.Text  =  "30～50 职业生涯规划";
    }
}
```

运行内容页 property.aspx，效果如图 5-9 所示。





图 5-9　根据用户选择提供不同主题

注意：如果想访问母版页中的成员，还需要为当前内容页添加指令＜％@ MasterType＞对母版页进行强类型化。如＜％@ MasterType VirtualPath ="～/getMasterValue/MainMaster.master" ％＞，其中 VirtualPath 用来设置母版页的位置。

2）使用 FindControl()方法

运行时，母版页将于内容页合并成一个页面，因此在内容页中可以访问母版页中的控件。但由于先加载内容页，后加载母版页，所以不能使用常规的方法来访问，要通过 this.Master.FindControl(服务器控件 ID)方法找到母版页中的控件，页面 FindControl.aspx 的隐藏 cs 文件中关键代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    RadioButtonList radioButtonList = (RadioButtonList)this.Master.FindControl("rblRole");
    if (radioButtonList.SelectedValue .Equals ("0"))
    {
        this.lblText.Text = "高校 BBS 求职信息";
    }
    else if (radioButtonList.SelectedValue.Equals("1"))
    {
        this.lblText.Text = "30～50 职业生涯规划";
    }
}
```

因 FindControl()方法返回值是 Control 类型，因此取得的控件需要进行类型转换。

提示：如果要访问母版页中的公共方法，可以通过 Master 直接调用。

5.1.4　站点导航

在网站制作中，站点导航是很常见的模块。早期由于没有一种简便的方式，产生了很多中导航的方式，比如硬编码、包含文件等。其基本方式就是在页面放置相关的超链接，以达到导航的功能。这些方式的缺点是不宜维护、导航不能集中管理。为了解决这个问题，ASP.NET 提供了站点导航的一种简单方法，即使用站点导航控件 SiteMapPath、TreeView、Menu 等，这三个导航控件都能够快速的建立导航，并且能够调整相应的属性为导航控件进行自定义，接下来将重点介绍 SiteMapPath、TreeView 控件。

1. 站点地图

为了使用 ASP.NET 的导航功能，必须有一种标准的方法描述站点中的每个页面。这个标准不仅包含每个网页的名称，还应该能够表明它们的层次结构关系，该关系是由页面的实际物理关系决定的。比如，要做出售软件开发相关书籍的网站，就可能是这样一个导航路径：软件开发→开发语言→C＃。

在 ASP.NET 中，有一个叫作站点地图的 XML 文件包含这些信息。站点地图的文件名默认是 web.sitemap，默认放置于应用程序的根目录，下面这段代码就是网站项目 SiteMapPath（位于 chapter05 目录下）的地图的代码：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< siteMap xmlns = "http://schemas.microsoft.com/AspNet/SiteMap - File - 1.0">
  < siteMapNode url = "Default.aspx" title = "软件开发" description = "">
    < siteMapNode url = "Default2.aspx" title = "开发语言" description = "">
      < siteMapNode url = "csharp.aspx" title = "C＃" description = ""/>
      < siteMapNode url = "vbdotnet.aspx" title = "VB.NET" description = ""/>
    </siteMapNode >
  </siteMapNode >
</siteMap >
```

节点描述如下：

- siteMap——根节点，一个站点地图只能有一个 siteMape 元素。
- siteMapNode——对应于页面的节点，一个节点描述一个网页。各节点间的层次关系是由该节点的结束标签(</siteMapNode>)决定的。
- title——页面相关的标题概述(与网页头部的 title 标记没有任何关系)。
- url——页面在解决方案中的位置，Url 不是必需的，如果没有设置 url 代表此节点不用来导航页面。
- description——页面相关的说明性文本。

编写站点导航地图的注意事项：

- 站点地图根节点为<siteMap>元素，每个文件有且仅有一个根节点。
- <siteMap>下一级有且仅有一个<siteMapNode>节点，该<siteMapNode>通常用来表示站点的首页。
- <siteMapNode>下面可以包含多个新的<siteMapNode>节点。
- 在站点地图中，同一个 URL 仅能出现一次。

说明：站点地图文件指向的页面关系是逻辑关系，而不是存储位置间的关系，所以具有很大的灵活性。

2. SiteMapPath 控件

SiteMapPath 控件可以为站点提供"面包屑导航"的功能，该控件根据 Web. sitemap 定义的数据自动显示当前页面的位置，并以链接的形式显示返回主页的路径。在站点的设计中，我们需要给用户提供一个方便的路径，如图 5-10 所示的网易新闻导航。

网易 > 新闻中心 > 国内新闻 > 正文

图 5-10　网易新闻的面包屑导航

这是典型的面包屑导航。该导航显示了从站点的首页到当前页面之间的路径。

SiteMapPath 控件使用起来非常方便，它使用站点地图作为控件的数据源。所以要使用该控件，首先要有站点地图，将刚才创建的站点地图，添加于站点地图中对应 Default. aspx、csharp. aspx、vbdotnet. aspx 等页面；然后直接将 SiteMapPath 控件拖入 csharp. aspx 页面中，该控件会自动读取根目录下的 web. sitemap 文件，效果如图 5-11 所示。

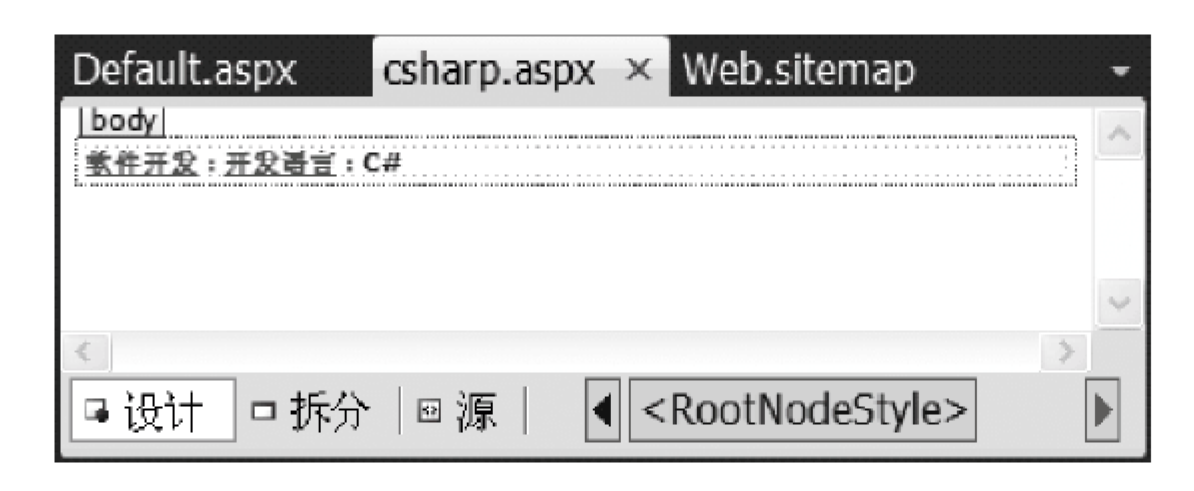


图 5-11　读取站点地图的 SiteMapPath 控件

SiteMapPath 控件使用起来非常方便，图 5-11 中链接的文字是通过 title 属性设置的，链接的提示信息是通过 description 属性设置的，表 5-1 列出了 SiteMapPath 控件的常用属性。

表 5-1　SiteMapPath 控件的常用属性

属　性	说　明
PathSeparator	控制分隔符的样式，可以通过编辑模板更改分隔符为任意样式
PathDirection	要呈现的路径方向，可选值有：RootToCurrent，这是默认值，表示从根级显示到当前级；CurrentToRoot，表示从当前页显示到根级
ParentLevelsDisplaye	要显示的父节点的数目，默认为－1，表示显示所有父节点

默认情况下，导航的分隔符是“＞”，如果需要使用其他字符，可以通过修改 PathSeparator 属性来达到。如果希望设置为图片，则需要使用分隔符模板，可以像图 5-12 那样编辑分隔符模板。



图 5-12　读取站点地图的 SiteMapPath 控件

注意：只有在站点地图中列出的页才能在 SiteMapPath 控件中显示导航信息。如果将 SiteMapPath 控件放置在站点地图中未列出的页上，该控件将不会向客户端显示任何信息。

说明：如果设置了分隔符属性又添加了分隔符模板，则显示时以模板为准。

5.1.5　TreeView 控件

TreeView 控件由一个或多个节点构成，树状结构中的每一项都称为“节点”。表 5-2 列出了三种不同的节点类型。

表 5-2　TreeView 控件的节点类型

节 点 类 型	说　　明
根节点	没有父节点,但具有一个或多个子节点的节点
父节点	具有一个父节点,且有一个或多个子节点的节点
叶节点	没有子节点的节点

TreeView 控件的应用相当普及,它以树状结构显示分层数据,如 Windows 的资源管理器左侧的文件目录就是一个相当经典的 TreeView 控件的应用例子,如图 5-13 所示。

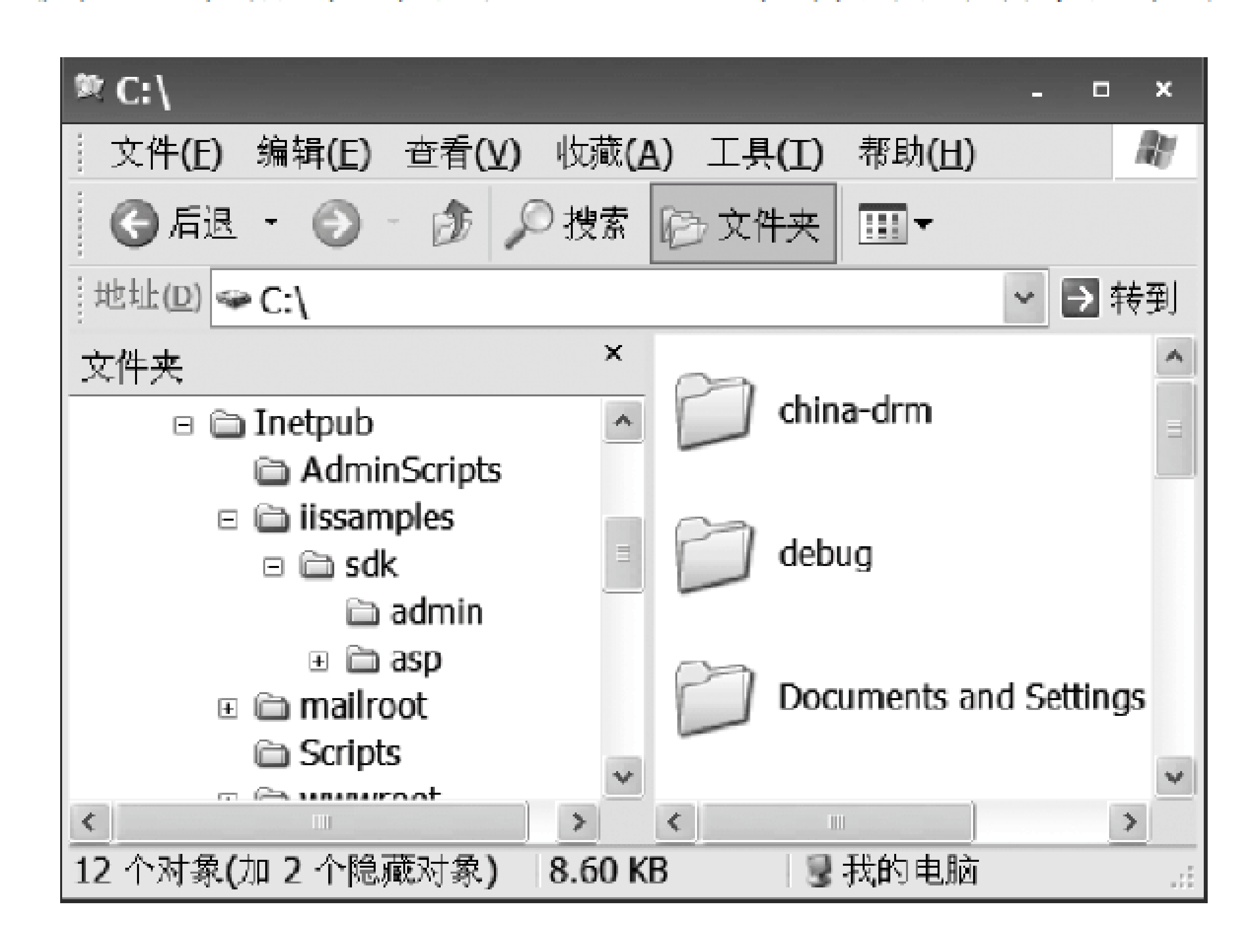


图 5-13　资源管理器的树状目录

使用传统的方式编写属性导航,需要复杂而庞大的代码量。使用 ASP.NET 中的 TreeView 导航控件,可以像 SiteMapPath 控件通过设置属性即可显示树状导航。

TreeView 的使用方式分为两种:视图中编辑和动态添加。

1. 视图中编辑

对于固定且少量数据可以采用在设计视图中编辑节点的方式,过程如下。

(1) 在"设计"视图中,右击 TreeView 控件,在弹出的快捷菜单中选择"编辑节点"命令。

(2) 在"TreeView 节点编辑器"对话框中的"节点"下,可以选择"添加根节点"、并且可以选中某个节点为其"添加子节点"和"移除节点"操作。

(3) 在"TreeView 节点编辑器"对话框中的"属性"列表框中可以对当前选中的节点进行编辑,如图 5-14 所示。

TreeView 的节点 TreeNode 常用属性如表 5-3 所示。

表 5-3　TreeNode 的常用属性

属　　性	说　　明
ChildNodes	获取 TreeNodeCollection 集合,该集合包含当前节点的第一级子节点
Depth	获取节点的深度
ImageUrl	获取或设置节点旁显示的图像的 URL
NavigateUrl	获取或设置单击节点时导航到的 URL

续表

属　性	说　明
Parent	获取当前节点的父节点
Text	获取或设置为 TreeView 控件中的节点显示的文本
Value	获取或设置用于存储有关节点的任何其他数据(如用于处理回传事件的数据)的非显示值
ValuePath	获取从根节点到当前节点的路径



图 5-14　资源管理器的树状目录

(4) 还可以为 TreeView 选定样式,右击 TreeView 控件,在弹出的快捷菜单中选择“自动套用格式”命令,在打开的“自动套用格式”对话框中,可以看到 TreeView 控件内置了很多可选择的样式,至此,我们没有编写一行代码就轻松实现了树状导航功能,如图 5-15 所示(见网站项目 ch05 下的 TreeViewDemo.aspx)。

注意:*在实际开发中,不推荐使用此种方式,其缺点就是扩展性差,不易维护,设想当数据复杂且数据量较大时,其工作量可想而知。项目中是以编程方式动态添加 TreeView 节点数据。*

图 5-15　在视图中编辑 TreeView 效果

2. 动态添加

视图中编辑的方式添加 TreeView 节点数据的缺点是不易维护,对于数据的处理不够灵活。动态添加节点方式根据数据的读取方式的不同又可以分为数据绑定方式和递归法动态添加节点方式。

1) 数据绑定方式

通常将菜单内容集中存储,如站点地图或 XML 文件等,然后通过使用数据源控件和

TreeView 控件关联来展示站点的导航层次结构。

（1）使用站点地图。

这里使用已经创建好的站点地图，首先创建母版页 MasterPage.master，将 TreeView 控件拖入设计视图并设置数据源，如图 5-16 所示。



图 5-16　将 TreeView 拖入页面

选择“新建数据源”后，页面弹出“数据源配置向导”对话框，可以看到 TreeView 控件可以使用两种形式的数据源，如图 5-17 所示。

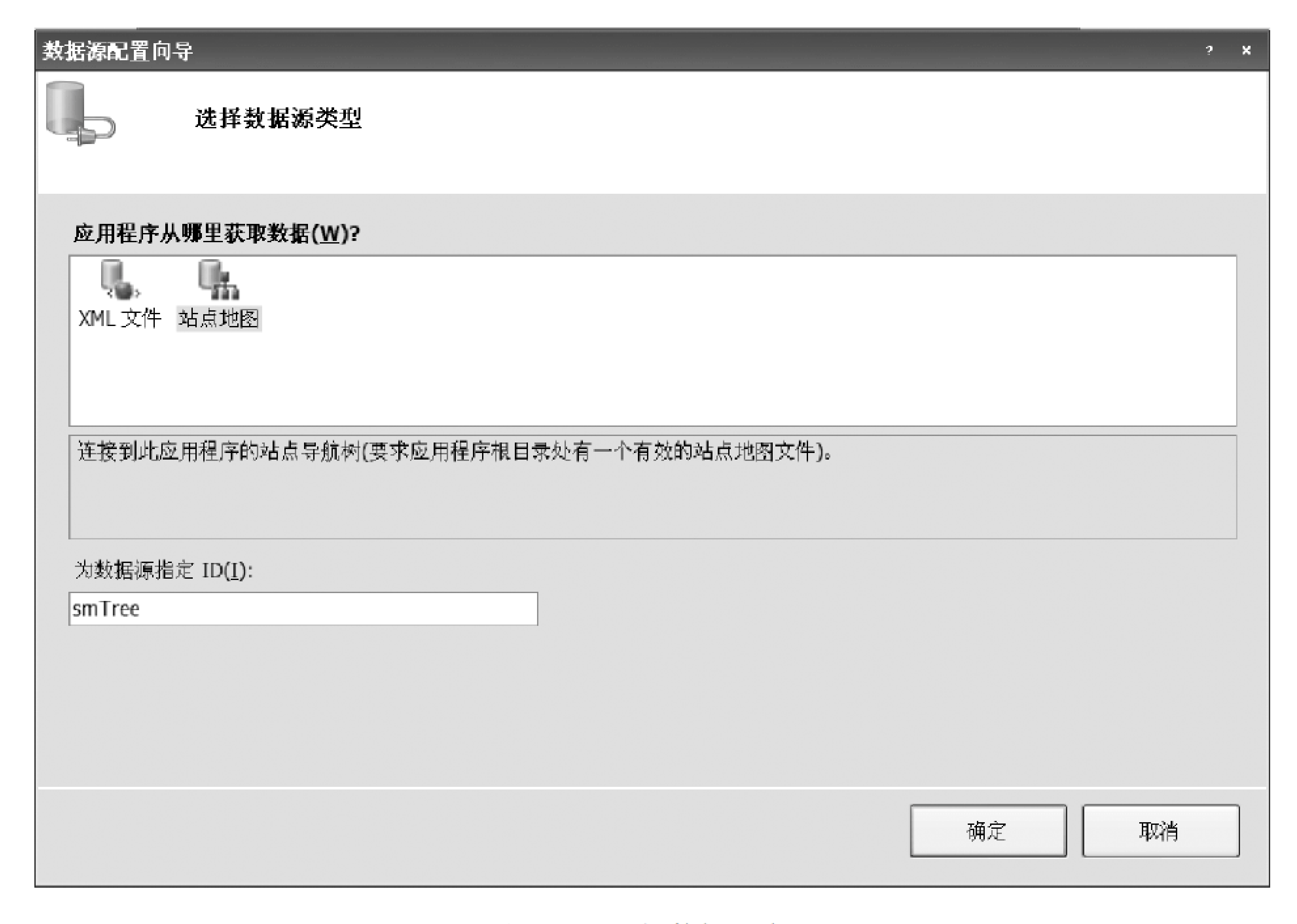


图 5-17　选择数据源类型

选择“站点地图”选项，并为数据源指定ID，此处命名为smTree，单击“确定”按钮后，树状导航就显示了，效果如图5-18所示。

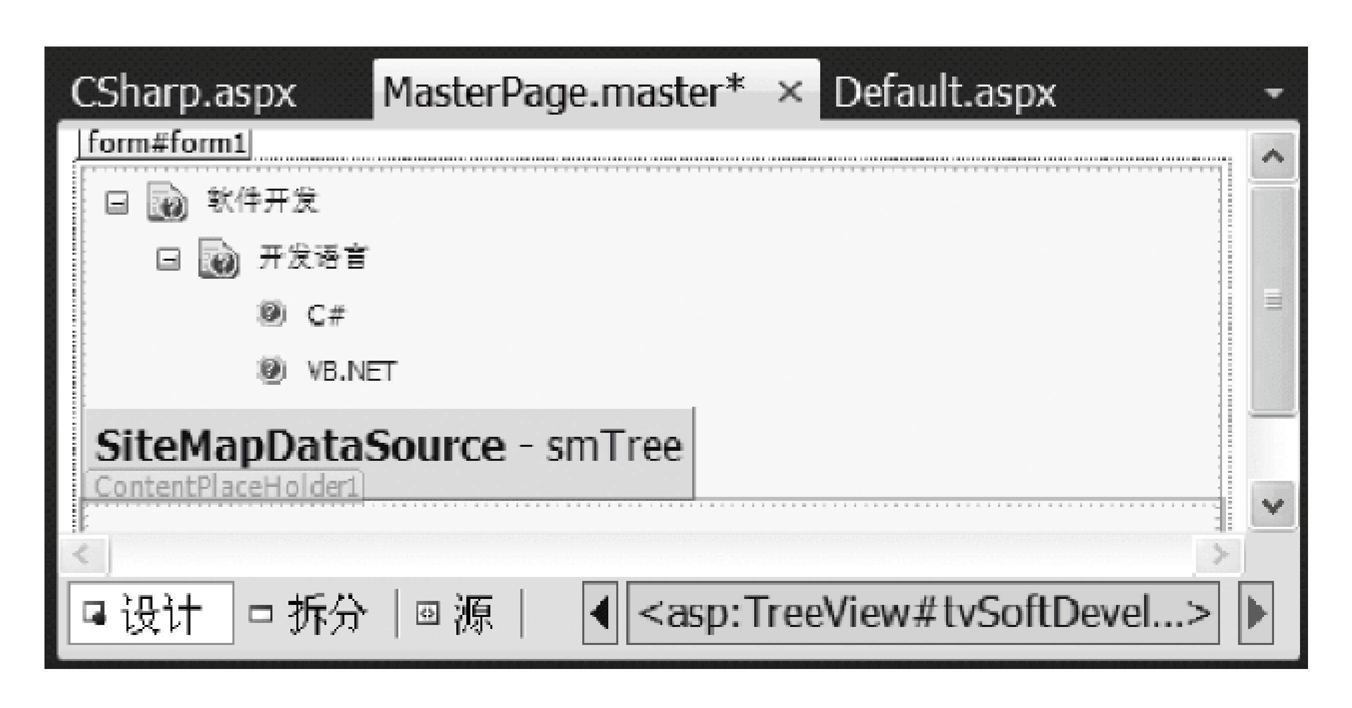


图5-18　使用站点地图方式实现树状导航

(2) 使用XML文件。

TreeView控件除了与站点地图绑定外，同样还可以与XML文件进行绑定，步骤如下。

① 编写XML文件。

首先编写一个XML格式的文件，该XML文件的内容参考站点地图文件中的内容。与站点地图相比，这里需要的XML文件没有那么多限制条件，甚至从站点地图中复杂部分代码过来就可以，只要符合XML标准即可。

结合需求，设置XML文件的内容如下：

```
<?xml version = "1.0" encoding = "utf - 8"?>
< siteMapNode url = "Default.aspx" title = "软件开发" description = "">
  < siteMapNode url = "Default2.aspx" title = "开发语言" description = "">
     < siteMapNode url = "csharp.aspx" title = "C# " description = ""/>
     < siteMapNode url = "vbdotnet.aspx" title = "VB .NET" description = ""/>
  </siteMapNode >
</siteMapNode >
```

这里将该XML文件保存为xmlTree.xml。

② 设置数据源。

在页面中拖入TreeView控件，然后设置TreeView的数据源为XML格式数据源，这两步和使用站点地图文件作为数据源的方式一样，只要选择“XML文件”作为数据源并为数据源指定ID即可。单击“确定”按钮后，弹出配置数据源对话框，如图5-19所示。

其中，“数据文件”项用于设置XML文件的路径，可单击后面的“浏览”按钮，选择需要的XML文件，单击“确定”按钮后，页面显示如图5-20所示的结果。

③ 编辑数据绑定。

很显然，TreeView并未识别我们的意图：我们希望siteMapNode元素的title属性作为显示字段，url属性作为链接。那么需要做一些设置，如图5-21所示，单击智能提示中的“编辑TreeNode数据绑定”，打开“TreeView DataBindings编辑器”对话框。

在该编辑器对话框中，添加将要绑定的节点，然后从右侧设置绑定的元素，如图5-22所示。



图 5-19 “配置数据源”对话框



图 5-20 未编辑绑定时 TreeView 的显示效果



图 5-21 编辑数据绑定



图 5-22 数据绑定编辑器

其中，常见的数据绑定属性如表 5-4 所示。

表 5-4　TreeView 数据绑定的常用属性

属　　性	说　　明
TextField	设置显示的文字所绑定的字段或元素
NavigateUrlField	设置链接对应的字段或元素

④ 设置格式。

至此，TreeView 就可以正常显示了，还可以通过“自动套用格式”选择显示样式，运行效果如图 5-23 所示。



图 5-23　TreeView 绑定 XML 文件后的运行效果

2）编程实现递归法动态添加节点

在实际项目开发中，菜单项作为数据的一部分通常被保存在数据库中，有专门的数据维护人员通过内部系统管理平台来操作维护菜单项，而菜单的深度也会随着网站的用户需求不断完善。显然不管使用站点地图或是 XML 文件方式维护数据都很困难。下面通过编程实现递归法动态添加节点、设置属性。从而实现数据与代码的分离，提高数据的可维护性。

图 5-24 是一个企业 OA 中的系统菜单表。其中 NodeId 字段为每个菜单项的编号，DisplayOrder 字段为显示顺序，ParentNodeId 字段为父节点的编号。如何按照图 5-24 中的数据结构将数据读取到 TreeView 中呢？

表 - dbo.SysFun　摘要

NodeId	DisplayName	NodeURL	DisplayOrder	ParentNodeId
101	人事管理		1	0
102	日程管理		2	0
103	文档管理		3	0
104	消息传递		4	0
105	系统管理		5	0
106	考勤管理		6	0
101001	机构信息	PersonManage/BranchManage.aspx	1	101
101002	部门信息	PersonManage/DepartManage.aspx	2	101
101003	员工管理	SysManage/UserManage.aspx	3	101
102001	我的日程	ScheduleManage/PersonSchedule/PersonSchedule.aspx	4	102
102002	部门日程	ScheduleManage/DepartSchedule/DepartSchedule.aspx	5	102
102003	我的便签	ScheduleManage/PersonNote/PersonNote.aspx	6	102
103001	文档管理	ScheduleManage/File/FileManage/FileManage.aspx	7	103
103002	回收站	ScheduleManage/File/RecycleBin.aspx	8	103
103003	文件搜索	ScheduleManage/File/FileSearch.aspx	9	103
104001	消息管理	Message/MessageManage/MessageManage.aspx	10	104
104002	信箱	Message/MailBox/MailBox.aspx	11	104
105001	角色管理	SysManage/RoleManage/RoleManage.aspx	12	105
105002	登录日志	SysManage/LoginLog.aspx	13	105
105003	操作日志	SysManage/OperateLog.aspx	14	105
105004	菜单排序	SysManage/MenuAdjust.aspx	15	105
106001	员工签到、签退	ManualSign/ManualSign.aspx	16	106
106002	考勤历史查询	ManualSign/ManualSignSearch.aspx	17	106
106003	考勤统计	ManualSign/SignStatistic.aspx	18	106
NULL	NULL	NULL	NULL	NULL

图 5-24　OA 系统菜单表中的数据信息

注意：取 ParentNodeId 为 0 的节点作为一级节点添加到 TreeView 中。根据每个一级节点的 NodeId 的值找到与其相等的 ParentNodeId 的值，如果找到就把找到的节点作为子节点添加到该一级节点。

说明：我们看到的系统菜单表只有两级，但随着业务的拓展该表中的菜单项级别会越来越高，为了提高扩展性，采用递归方法进行无限级添加。

关键代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        this.tvMenu.Dispose();
        GetDataToTable();                                    //将数据库中的数据填充到 DataSet
        InitTreeByDataTable(this.tvMenu.Nodes, "0");         //使用递归方法动态添加节点
    }
}
/// <summary>
/// 使用递归方法动态添加节点(DataTable 方式实现)
/// </summary>
/// <param name="tnc">父节点</param>
/// <param name="parentId">父节点 Id</param>
private void InitTreeByDataTable(TreeNodeCollection tnc,string parentId)
{
    DataView dv = new DataView();                            //动态视图方便筛选
    TreeNode tnNode;
    dv.Table = ds.Tables[0];                          //全局的 DataSet,对应系统菜单表 SysFun
    dv.RowFilter = "ParentNodeId=" + parentId;
    foreach (DataRowView drv in dv)
    {
        tnNode = new TreeNode();
        tnNode.Value = drv["NodeId"].ToString();
        tnNode.Text = drv["DisplayName"].ToString();
        tnNode.NavigateUrl = drv["NodeURL"].ToString();
        tnc.Add(tnNode);
        InitTreeByDataTable(tnNode.ChildNodes, tnNode.Value);   //递归调用
    }
}
```

上述代码中，TreeNodeCollection 表示 TreeView 控件中的 TreeNode 对象的集合。

这里默认 ParentNodeId 为 0 的是一级节点，从一级节点开始，利用视图，筛选出 ParentNodeId 值与此节点的 NodeId 值相等的节点集合，遍历该节点集合并将其每个节点作为子节点进行添加。循环中调用的 InitTreeByDataTable()方法每次将当前节点作为父节点传入，形成递归。递归是一种重要的编程技术，它可以实现让一个函数从其内部调用其自身。使用递归法时要特别注意退出条件。

切换到页面 Default.aspx（在文件夹 RecursionTreeView 中）的设计视图模式，为 TreeView 选择“自动套用格式”并运行代码，效果如图 5-25 所示。

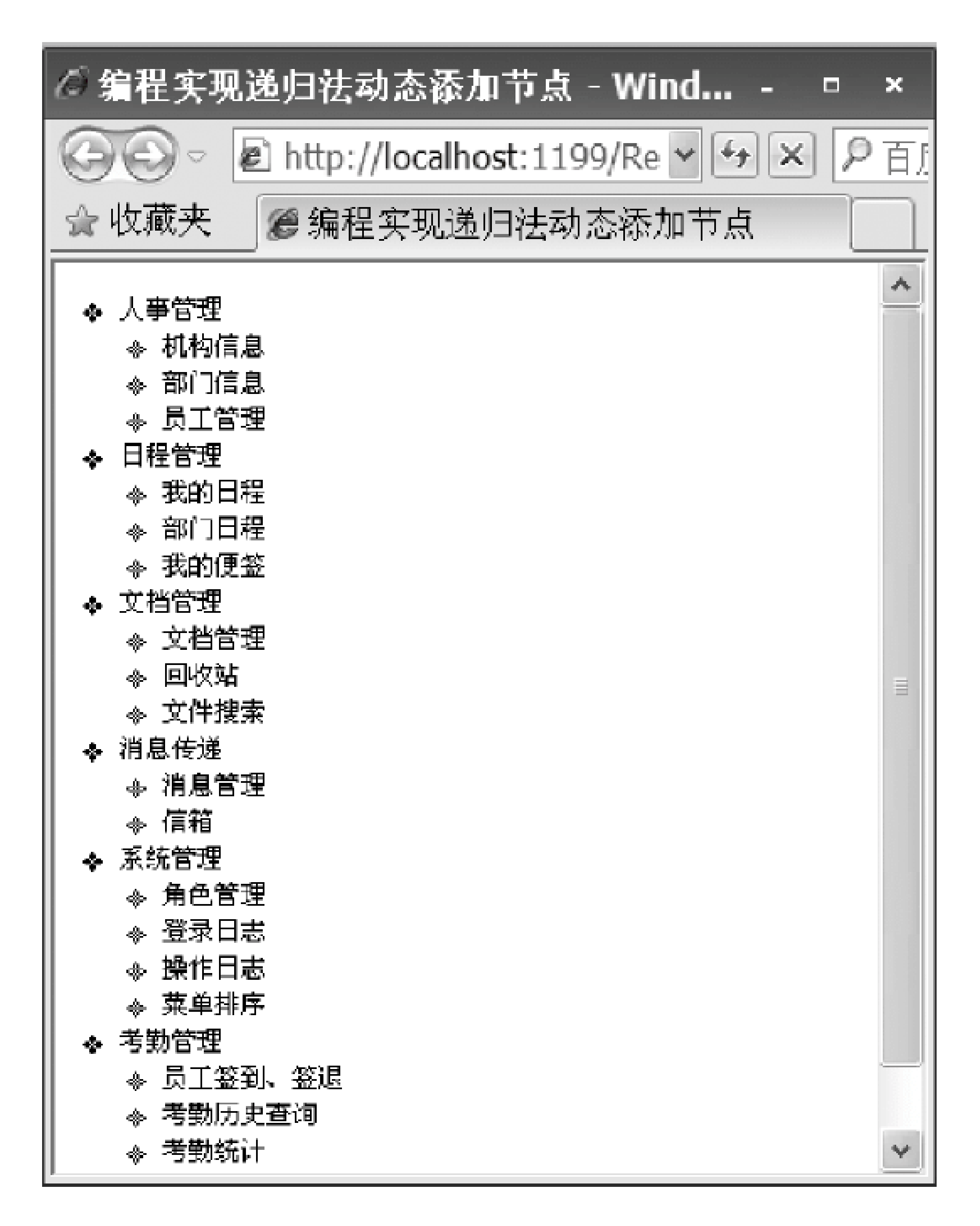


图 5-25　编程实现递归法动态添加 TreeView 节点数据

但这种方式添加节点也存在缺点，对于数据量大的情况，若一次性将菜单数据全部都加载到页面中会降低网站的性能，因此可以采用当用户每单击一个一级节点，调用相应方法获取该级节点下全部子节点的方法来提升网站的性能。

5.2　单元任务

任务 5-2-1　使用母版页搭建“新知书店”管理端页面框架

【任务描述】

- 使用母版页搭建如图 5-26 所示的“新知书店”页面框架。
- 在母版页中显示“管理员，您好！”，在内容页中显示“欢迎使用新知书店管理端！”。

【任务实施】

（1）运行 Visual Studio 2010，在 VS2010 菜单栏中选择“文件”→“新建网站”命令，打开“新建网站”对话框，在文件夹 rw5-2-1 下创建名为 Web 的网站项目，将解决方案另存为 BookShop 保存在文件夹 rw5-2-1 下。

（2）通过右击网站项目 Web，新建文件夹 Admin，在 Admin 文件夹下创建文件夹 Css 和 Images 用于存放新知书店管理端相关资源和代码文件。

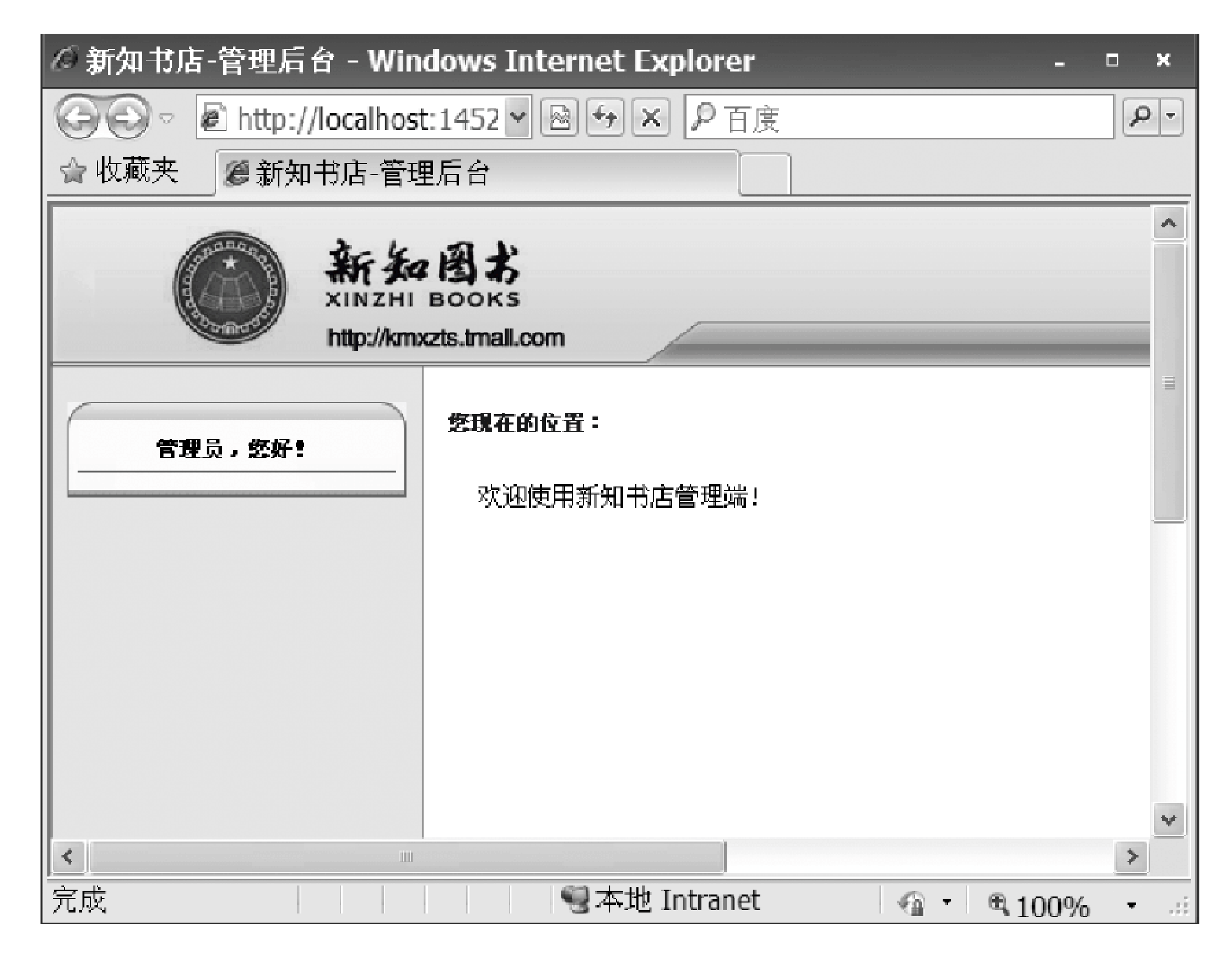


图 5-26 “新知书店”管理端页面框架

（3）在 Admin 文件夹下添加母版页 Admin.master，为母版页设计布局并添加样式，此处样式文件 admin.css 在文件夹 Css 下，样式文件代码如下：

```
/* CSS Document */
*{ margin:0; padding:0;}
body,select{ font:14px/20px 宋体;}
table{ border-collapse:collapse;}
img{ border:0; vertical-align:middle;}
ul{ list-style:none;}
a{ color:#0072A7; text-decoration:underline;}
a:hover{ color:#f00;}
.black, .black a, a.black{ color:#333; text-decoration:none;}
.red, .red a, a.red{ color:#f00;}
.white, .white a, a.white{ color:#fff;}
.f_left{ float:left;}
.f_right{ float:right;}
.del{ text-decoration:line-through;}

/*后台管理主界面*/
#header{ margin:0 auto; width:1003px; height:92px;}
#main{ margin:0 auto; width:1003px; overflow:hidden;}
/*左侧菜单*/
#opt_list, #opt_area{ margin-bottom:-10000px; padding-bottom:10000px;}
/*左侧菜单*/
#opt_list{ float:left; width:210px; height:590px; border:solid #ACB9C1; border-width:0 1px;
```

```
            background:#EAEFF1 url(../images/admin_menu_t.gif) no-repeat center 20px;
            padding-top:35px;_background-position-x:8px;}
#opt_list h1{ width:181px; text-align:center; border-bottom:1px solid #5F8AB4; margin:0
auto 10px;
             font:600 12px/24px 宋体;}
#subnav{  margin:0 auto; width:191px; border:solid #0C9EE1; border-width:0 1px 1px;
            background:#FCFCFC url(../images/admin_menu_b.gif) repeat-x 0 bottom;
            padding-bottom:3px; font-size:14px; font-family:@宋体;}
/*右侧内容*/
#opt_area{ float:right; width:760px;}
/*通用部分*/
#breadcrumb{ margin:20px; font:600 12px/24px 宋体;}
#breadcrumb a{text-decoration:underline}
.data_table{ margin:0 auto; border:2px solid #999; width:758px;}
.data_table td, .data_table th{ border:groove #A29C9E; border-width:0 1px; font:14px/24px
宋体;}
.data_table th{ background:url(../images/date_th.gif) repeat-x; line-height:30px;
                          border-bottom:1px solid #fff; font-weight:600; color:Black}
.data_table td{ text-align:center; padding:10px 6px 8px 10px;}
.data_table td.name{ text-align:left;}
.opt_action{ margin:30px 0 0 20px; font:14px/20px 宋体;}
.opt_action select{ width:100px; font:14px Tahoma;}
.opt_action input{ height:24px; padding:0 6px;}
.pages{ height:40px; text-align:right; padding:20px 0 0 100px;}
.pages td{ border-width:0px;}
.table_edit{ background-color: #d7effb; width: 758px; border:solid 1px #0f9fde;}
.table_edit th{ border-right:solid 1px #0f9fde; border-bottom:solid 1px #0f9fde;
width:100px;}
.table_edit td{ border-right:solid 1px #0f9fde; border-bottom:solid 1px #0f9fde; }
.table_detail{ background-color: #d7effb; width: 758px; border:solid 1px #0f9fde;}
.table_detail th{ border-right:solid 1px #0f9fde; border-bottom:solid 1px #0f9fde;
width:100px;}
.table_detail td{ border-right:solid 1px #0f9fde; border-bottom:solid 1px #0f9fde; }
```

在母版页 Admin.master 的<head></head>标记对之间写入导入样式文件的代码，在主体部分编写布局的代码如下：

```
<head runat="server">
    <title>新知书店-管理后台</title>
    <link href="CSS/admin.css" rel="stylesheet" type="text/css" />
</head>
<body>
    <form id="Form1" runat="server">
    <div id="header"><img src="images/admin_top.gif" alt="" /></div>
    <div id="main">
        <div id="opt_list">
```

```
                <h1>管理员,您好!</h1>
                <div id="subnav" >
                </div>
            </div>
            <div id="breadcrumb" class="black">
                  您现在的位置:
            </div>
            <div id="opt_area">
                <asp:ContentPlaceHolder ID="cphAdmin" runat="server">
                </asp:ContentPlaceHolder>
            </div>
        </div>
        </form>
    </body>
```

(4) 添加基于母版页 Admin.master 的内容页 Default.aspx,并在内容页的 Content 控件中添加"欢迎使用新知书店管理端!",注意内容页中的 ContentPlaceHolderID 属性要与母版页中的 ID 属性相同,代码如下:

```
<asp:content id="Content1" contentplaceholderid="cphAdmin" runat="Server">
     欢迎使用新知书店管理端!
 </asp:content>
```

任务 5-2-2　实现"新知书店"管理端"面包屑"导航功能

【任务描述】

- 编写站点地图,实现如图 5-27 所示的层次结构。

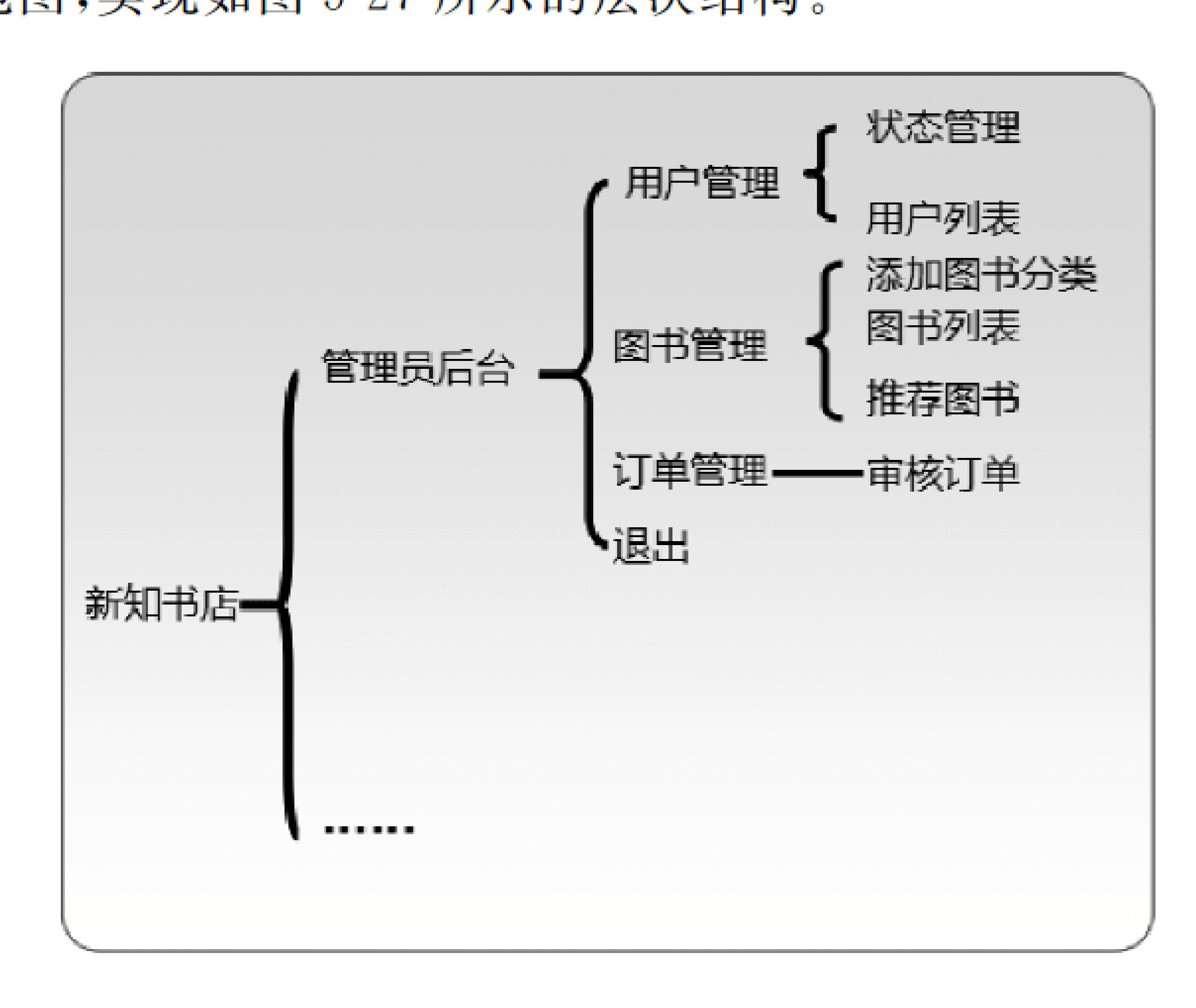


图 5-27　"新知书店"管理端页面层次结构

- 在任务 5-2-1 的基础上实现"新知书店"管理端"面包屑"导航功能,效果如图 5-28 所示。



图 5-28　“新知书店”管理端首页

【任务实施】

（1）在文件夹 rw5-2-2 下创建网站项目 Web，解决方案名称为 BookShop，将任务 5-2-1 的站点目录下的文件及文件夹复制至新创建的网站项目 Web 下。

（2）通过右击网站项目 Web，添加“站点地图”文件 web.sitemap，并根据如图 5-27 所示的层次结构编写代码如下。

```
<?xml version = "1.0" encoding = "utf - 8" ?>
< siteMap xmlns = "http://schemas.microsoft.com/AspNet/SiteMap - File - 1.0" >
    < siteMapNode Id = "" url = "~\Admin\Default.aspx" title = "管理员后台" description = "">
      < siteMapNode url = "" title = "用户管理" description = "">
        < siteMapNode url = "~\Admin\UserList.aspx" title = "管理用户" description = "" />
        < siteMapNode url = "~\Admin\UserStateManage.aspx" title = "状态管理" description = "" />
        < siteMapNode url = "~\Admin\UserDetails.aspx"  title = "修改用户资料"  description = "" />
      </siteMapNode >
      < siteMapNode url = "" title = "图书管理" description = "">
        < siteMapNode url = "~\Admin\CategoryManage.aspx" title = "添加图书分类" description = "" />
        < siteMapNode url = "~\Admin\BookCategory.aspx" title = "为书籍分类" description = "" />
        < siteMapNode url = "~\Admin\BookDetail.aspx" title = "图书详细信息" description = "" />
        < siteMapNode url = "~\Admin\BookList.aspx" title = "图书列表" description = "" />
        < siteMapNode url = "~\Admin\RecomBookList.aspx" title = "推荐图书" description = "" />
      </siteMapNode >
      < siteMapNode url = "" title = "订单管理" description = "">
        < siteMapNode url = "~\Admin\OrderList.aspx" title = "审核订单" description = "" />
        < siteMapNode url = "~\Admin\OrderDetail.aspx" title = "详细订单" description = "" />
      </siteMapNode >
      < siteMapNode url = "~\Membership\LoginOut.aspx" title = "退出" description = "管理员退出">
      </siteMapNode >
```

```
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

（3）打开母版页 Admin.master，添加控件 SiteMapPath，该控件会自动读取刚才在根目录下创建的站点文件 web.sitemap。

任务 5-2-3　实现“新知书店”管理端的菜单功能

【任务描述】

管理员菜单项较简单，是为了便于维护使用 XML 文件保存菜单，本任务实现如图 5-29 所示的管理员菜单功能。



图 5-29　“新知书店”管理端默认首页（含菜单）

【任务实施】

（1）以任务 5-2-2 基础，在 Admin 文件夹下创建 XML 文件 admin_menu.xml，用来存储菜单项，菜单项比较简单，参照如图 5-27 所示的层次结构设置菜单，代码如下。

```
<?xml version = "1.0" encoding = "utf - 8"?>
< siteRoot Id = "root" url = "" title = "管理员控制面板" description = "">
  < siteMapNode url = "" title = "用户管理" description = "">
    < siteMapNode url = "~\Admin\UserStateManage.aspx" title = "状态管理" description = "" />
    < siteMapNode url = "~\Admin\UserList.aspx" title = "用户列表" description = "" />
  </siteMapNode>
  < siteMapNode url = "" title = "图书管理" description = "">
    < siteMapNode url = "~\Admin\CategoryManage.aspx" title = "添加图书分类" description = "" />
```

```
      <siteMapNode url="~\Admin\BookList.aspx" title="书籍列表" description="" />
    </siteMapNode>
    <siteMapNode url="" title="订单管理" description="">
      <siteMapNode url="~\Admin\OrderList.aspx" title="审核订单" description="" />
    </siteMapNode>
    <siteMapNode url="~\Membership\LogOut.aspx" title="退出" description="管理员退出">
    </siteMapNode>
  </siteRoot>
```

（2）打开母版页 Admin.master，从工具箱中拖放一个 TreeView 控件至页面，单击右上方的箭头，弹出“TreeView 任务”对话框，在“选择数据源”下拉列表框中选择“新建数据源”选项，在弹出的“数据源配置向导”对话框中选择“XML 文件”，并指定数据源 ID 为 xdsAdmin，单击“下一步”按钮，选择第 1 步创建的数据文件 admin_menu.xml，进行布局调整，主要代码如下。

```
<asp:XmlDataSource ID="xdsAdmin" runat="server" DataFile="~/Admin/admin_menu.xml">
</asp:XmlDataSource>
<div id="header">
<img src="images/admin_top.gif" alt="" /></div>
<div id="main">
<div id="opt_list">
    <h1>管理员,您好!</h1>
    <div id="subnav" >
        <asp:TreeView ID="tvAdmin" runat="server" DataSourceID="xdsAdmin" ImageSet="Arrows"
            Width="191px">
            <DataBindings>
                <asp:TreeNodeBinding DataMember="siteMapNode" NavigateUrlField="url"
                                     TextField="title" />
                <asp:TreeNodeBinding DataMember="siteRoot" TextField="title" />
            </DataBindings>
            <ParentNodeStyle Font-Bold="False" />
            <HoverNodeStyle Font-Underline="True" ForeColor="#5555DD" />
            <SelectedNodeStyle Font-Underline="True" HorizontalPadding="0px"
                               VerticalPadding="0px"  ForeColor="#5555DD" />
            <NodeStyle Font-Names="Verdana" Font-Size="8pt" ForeColor="Black"
                       HorizontalPadding="5px" NodeSpacing="0px" VerticalPadding="0px" />
        </asp:TreeView>
    </div>
</div>
```

（3）打开母版页 Admin.master，添加控件 SiteMapPath，该控件会自动读取刚才在根目录下创建的站点文件 web.sitemap。

任务 5-2-4　搭建“新知书店”前台页面框架

【任务描述】

在任务 5-2-3 的基础上，使用母版页创建“新知书店”前台页面框架并添加默认页面，效

果如图 5-30 所示。



图 5-30 “新知书店”前台默认首页

【任务实施】

(1) 在网站项目 Web 下新建文件夹 Css 和 Images，用来存放“新知书店”前台相关图片资源和样式表文件，并将前台页面所需的图片资源和样式表文件复制到对应目录下。限于篇幅，样式表文件代码不予罗列。

(2) 右击网站项目 Web，创建母版页 common.master，为母版页设计布局，编写代码如下。

```
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
    <meta name="Robots" content="index,follow" />
    <title>新知书店-最方便的网上书店</title>
    <link href="CSS/global.css" rel="stylesheet" type="text/css" />
    <asp:ContentPlaceHolder ID="cphHeader" runat="server">
    </asp:ContentPlaceHolder>
</head>
<body>
    <form id="form1" runat="server">
    <div id="top">                                            //顶部
        <div class="status">
            您好,
```

```
            <asp:Literal ID = "ltlUser" runat = "server"><a href = "">【登录】</a><a href = "">
【免费注册】</a>
            </asp:Literal>
        </div>
        <div class = "member">
            <ul>
                <li><a href = "#"><img src = "Images/payVIP.gif" height = "18" alt = "开通
VIP" /></a></li>
                <li><a href = "#"><img src = "Images/payCoin.gif" height = "18" alt = "学币
中心" /></a></li>
            </ul>
        </div>
    </div>
    <div id = "header">                 //头部
        <div id = "logo"><img src = "Images/logo.gif" alt = "新知书店" /></div>
        <div id = "nav"><div id = "a_b_01"></div>
            <ul id = "mainnav">       //首页水平标签栏
                <li><a href = "Default.aspx">首页</a></li>
                <li><a href = "">购物车</a></li>
                <li><a href = "">收藏架</a></li>
                <li><a href = "#">购物流程</a></li>
                <li><a href = "#">在线客服</a></li>
                <li><a href = "#">积分兑换</a></li>
                <li><a href = "">管理入口</a></li>
                <li><a href = "#">帮助</a></li>
            </ul>
        </div>
    </div>
    <div id = "container">              //主体部分
        <! -- left content -- >
        <div id = "intro">              //主体部分左栏
            <div id = "basket">
                <a href = "#" runat = "server" id = "hrefShoppinCart">目前您的购物篮是空的</a>
            </div>
            <div id = "search">
                 <asp:TextBox ID = "txtKeyword" runat = "server" CssClass = "search_key">
</asp:TextBox>
                <asp:Button ID = "btnSearch" runat = "server" UseSubmitBehavior = "false"
                            CssClass = "search_sub" />
            </div>
            <div id = "alltype">
                <h1 class = "all_type black"><a href = "#">查看所有分类>></a></h1>
                <div id = "subnav">
                    <div id = "subnavbottom">  </div>
                </div>
                <! -- subnav end -- >
            </div>
            <! -- link start -- >
            <div id = "s_b_03">
                <a href = "#">CSDN 开发者大会</a><br />
```

```
                <a href = "#">.BET 俱乐部活动月</a></div>
            <! -- link end -->
        </div>
         //下面为主体部分右栏,预留给内容页的主体部分
        <asp:ContentPlaceHolder ID = "cphContent" runat = "server">
        </asp:ContentPlaceHolder>
    </div>
    <div id = "footer">                                    //底部
        <% -- 省略部分 -- %>
   </div>
    </form>
</body>
```

标明<a href="#">的在“新知书店”中不予实现功能,只提供显示。

(3) 在网站项目 Web 下基于母版页 common.master 创建内容页面 Default.aspx,运行效果如图 5-30 所示。

任务 5-2-5 实现“新知书店”前台页面站点导航功能

【任务描述】

在任务 5-2-4 的基础上,使用站点地图实现前台页面的站点导航功能,如图 5-31 所示,在 web.sitemap 中追加前台页面文件结构,文件结构如图 5-32 所示,在母版页中添加 SiteMapPath 控件。



图 5-31 “新知书店”前台默认首页中的导航效果

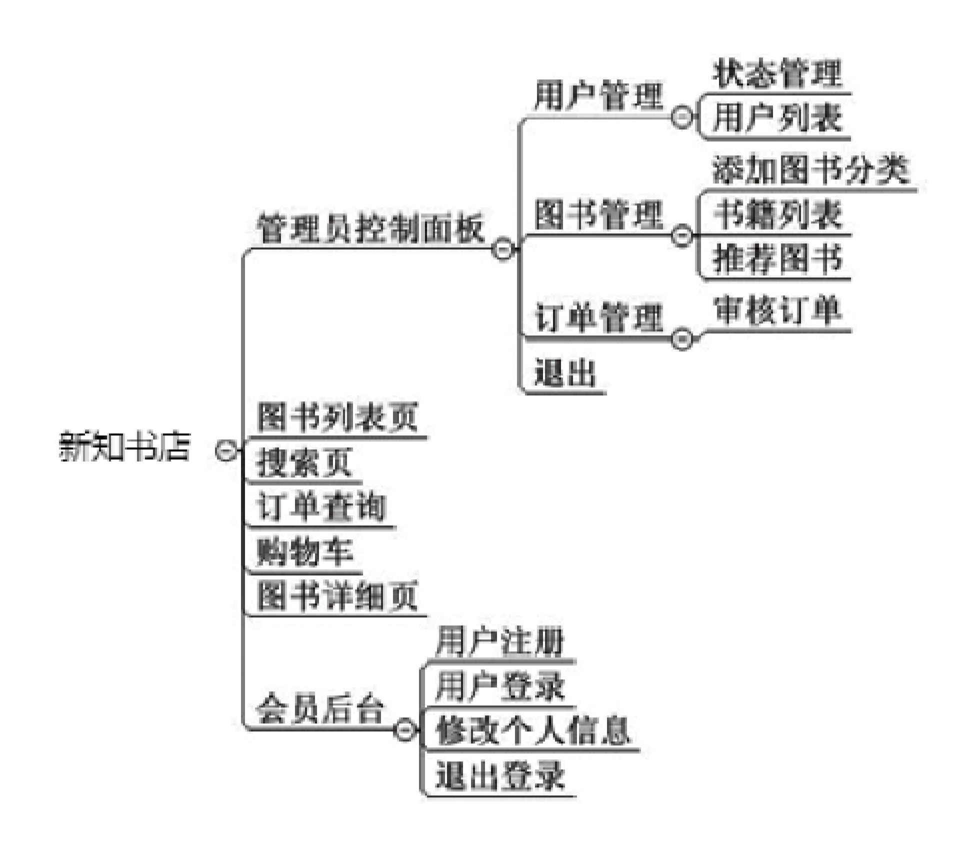


图 5-32　“新知书店”文件结构图

【任务实施】

(1) 该任务在任务 5-2-2 中曾实施过，较为简单，在任务 5-2-4 的基础上，把任务 5-2-2 中的站点文件 web.sitemap 复制到网站项目 Web 的根目录下，并追加如图 5-32 所示的前台页面文件结构，代码如下。

```
<?xml version = "1.0" encoding = "utf - 8" ?>
< siteMap xmlns = "http://schemas.microsoft.com/AspNet/SiteMap - File - 1.0" >
  < siteMapNode url = "～\Default.aspx" title = "新知书店"  description = "">
    < siteMapNode url = "BookList.aspx" title = "图书列表页"  description = ""/>
    < siteMapNode url = "Search.aspx" title = "搜索页"  description = ""/>
    < siteMapNode url = "" title = "订单查询"  description = "" />
    < siteMapNode url = "ShoppingCart.aspx" title = "购物车"  description = "" />
    < siteMapNode url = "～\BookDetail.aspx" title = "图书详细页"  description = "" />
    < siteMapNode Id = "" url = "" title = "会员后台" description = "">
      < siteMapNode url = "～\Membership\Register.aspx"  title = "用户注册"  description = "" />
      < siteMapNode url = "～\Membership\Login.aspx"  title = "用户登录"  description = "" />
 < siteMapNode url = "～\Membership\UserModify.aspx" title = "修改个人信息"  description = "" />
      < siteMapNode url = ""  title = "退出登录"  description = "" />
    </siteMapNode >
</siteMap >
```

(2) 从工具箱中拖放 SiteMapPath 控件至母版页 common.master 的相应位置，代码如下。

```
< div id = "breadcrumb" class = "black">
     您现在的位置:
   < asp:SiteMapPath ID = "smpRoad" runat = "server">
      < PathSeparatorStyle BackColor = "White" BorderColor = "White" />
   </asp:SiteMapPath >
</div >
```

(3) 将在任务 5-2-4 中创建的前台首页 Default.aspx 删除,根据母版页 common.master 重新创建页面 Default.aspx,即完成该任务的实施。

任务 5-2-6　实现"新知书店"前台页面菜单栏功能

【任务描述】

在任务 5-2-5 的基础上,结合 ADO.NET 读取数据表 Categories 中的数据并保存到泛型集合中,并使用 TreeView 的递归动态添加节点方法来展示图书分类信息,效果如图 5-31 所示。

【任务实施】

(1) 在任务 5-2-5 的基础上,创建网站项目 Web,保存在文件夹 rw5-2-6 中,在 web.config 中创建数据库连接字符串,代码如下。

```
<connectionStrings>
    <add name="BookShop" connectionString="Data Source=.;Initial
        Catalog=BookShopPlus;Integrated Security=True"/>
</connectionStrings>
```

(2) 右击网站项目 Web,选择"添加 ASP.NET 文件夹"->"App_Code",在该文件夹下创建类文件 Category.cs 和 SqlHelper.cs,Category.cs 是分类表数据表 Categories 对应的实体类,代码如下,SqlHelper.cs 封装了一些对数据进行操作的方法。

```
public class Category
{
    private int id;
    private string name = String.Empty;
    private int pId;
    private int sortNum;
    public Category(int id, string name)            //构造方法,根据 Id 获取名称 name
    {
        this.id = id;
        this.name = name;
    }
    public int Id                                   //分类编号
    {
        get { return this.id; }
        set { this.id = value; }
    }
    public string Name                              //类别名称
    {
        get { return this.name; }
        set { this.name = value; }
    }
    public int PId                                  //父类编号
```

```
        {
            get { return pId; }
            set { pId = value; }
        }
        public int SortNum                          //排序号
        {
            get { return sortNum; }
            set { sortNum = value; }
        }
    }
```

(3) 在母版页 common.master 中拖入 TreeView 控件至需要显示菜单的位置,代码如下。

```
<div id="alltype">
    <h1 class="all_type black"><a href="#">查看所有分类>></a></h1>
    <div id="subnav">
        <asp:TreeView ID="trvwCategory" runat="server"></asp:TreeView>
        <div id="subnavbottom"> </div>
    </div>
    <!-- subnav end -->
</div>
```

(4) 在母版页的后置代码文件 common.master.cs 中编写获取分类信息的泛型集合及绑定子节点的递归方法,代码如下。

```
protected void Page_Load(object sender, EventArgs e)
    {
        if (!Page.IsPostBack)
        {
            //动态加载分类
            this.LoadCategories();
        }
    }
    /// <summary>
    /// 绑定分类节点
    /// </summary>
    void LoadCategories()
    {
        List<Category> items = GetCategories();
        if (items.Count > 0)
            trvwCategory.Nodes.Clear();
        foreach (Category item in items)
        {
            if (item.PId == 0)
```

```
        {
            TreeNode node = new TreeNode(" " + item.Name, item.Id.ToString(),
                                         this.ResolveUrl("~/Images/arrow.gif"));
            node.NavigateUrl = "BookList.aspx?typeid=" + item.Id;
            this.trvwCategory.Nodes.Add(node);    //递归绑定子节点
            this.BindNode(items, node);
        }
    }

}

/// <summary>
/// 绑定子节点
/// </summary>
/// <param name="items">类别集合</param>
/// <param name="node">节点</param>
void BindNode(List<Category> items, TreeNode pNode)
{
    foreach (Category item in items)
    {
        if (item.PId == Int32.Parse(pNode.Value))
        {
            TreeNode node = new TreeNode(" " + item.Name, item.Id.ToString(),
                                          this.ResolveUrl("~/Images/arrow.gif"));
            node.NavigateUrl = "BookList.aspx?typeid=" + item.Id;
            pNode.ChildNodes.Add(node);
            BindNode(items, node);
        }
    }
}

/// <summary>
/// 获取分类信息并保存到泛型集合
/// </summary>
public List<Category> GetCategories()
{
    string strSql = "SELECT * FROM Categories ORDER BY SortNum ASC";
    List<Category> list = new List<Category>();
    DataSet ds = SqlHelper.ExecuteDataset(SqlHelper.ConnectionString,
                                                  CommandType.Text, strSql);
    if (ds.Tables.Count > 0)
    {
        DataTable dt = ds.Tables[0];
        foreach (DataRow row in dt.Rows)
        {
            Category category = new Category();
```

```
                    category.Id = (int)row["Id"];
                    category.Name = (string)row["Name"];
                    category.PId = (int)row["PId"];
                    category.SortNum = (int)row["SortNum"];
                    list.Add(category);
                }
            }
            return list;
        }
```

(5) 运行页面 Default.aspx,效果如图 5-33 所示。



图 5-33　“新知书店”前台默认页面菜单栏效果

5.3　项目实训

使用母版页搭建“博客系统”页面框架

【需求说明】

- 根据美工人员制作好的“博客系统”首页静态页面 MasterPage.html,搭建如图 5-34 所示的“博客系统”母版页 MasterPage.master(提示：将静态页面<body>中的代码放到母版页的<Form>标记当中)。
- 根据母版页创建“博客系统”注册内容页面 Register.aspx,如图 5-35 所示。



图 5-34　“博客系统”母版页效果



图 5-35　“博客系统”会员注册页效果(Register.aspx)

5.4　单元小结

本单元简单介绍了 CSS 样式控制，深入讲解了母版页的创建及与内容页的关系，通过示例演示了如何在内容页中获取母版页中的元素，阐述了站点导航技术及其相关控件的应

用，本单元知识体系如图 5-36 所示。

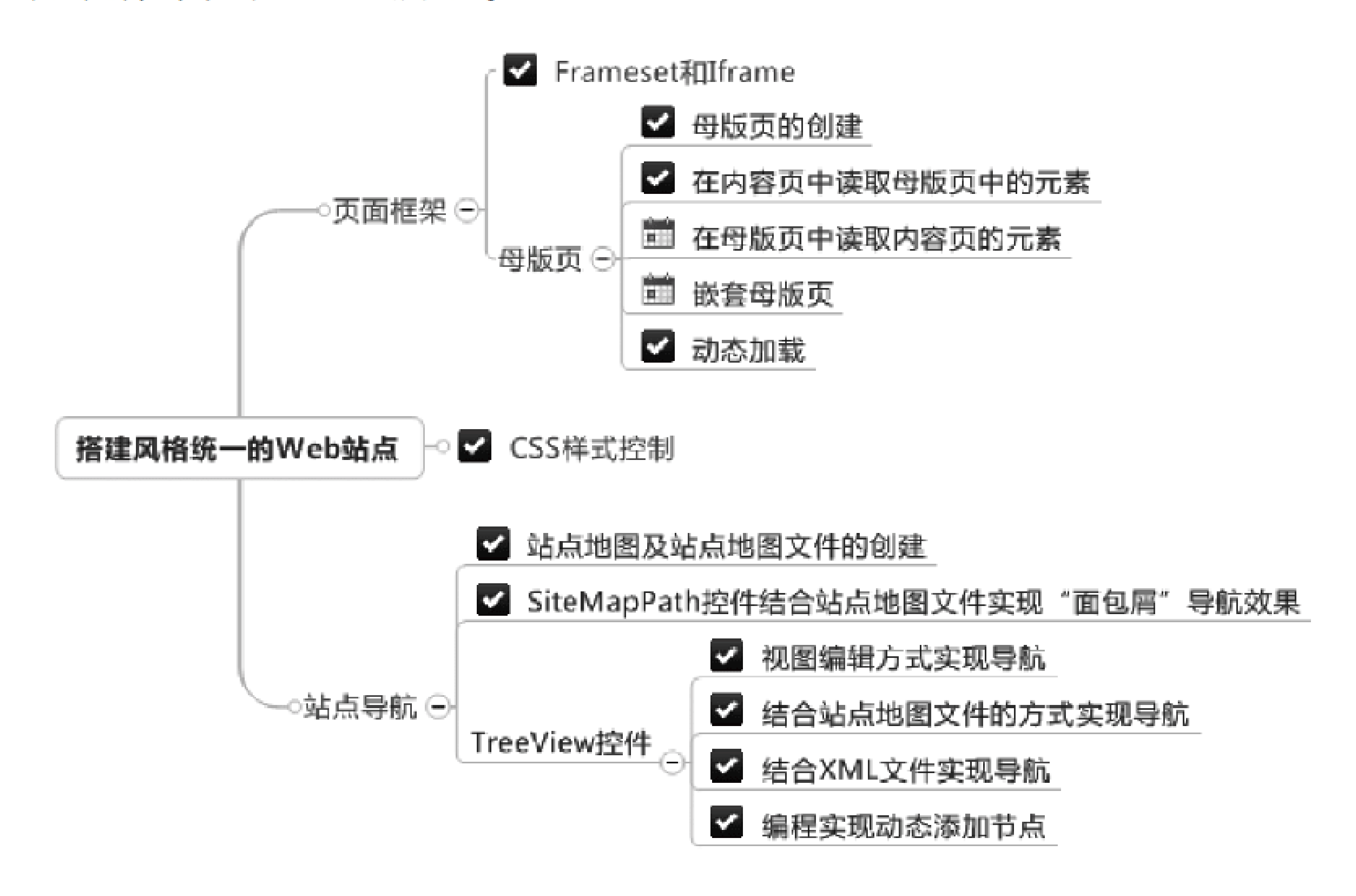


图 5-36　搭建风格统一的 Web 站点知识体系

5.5　单元练习题

一、选择题

1. 下面说法错误的是(　　)。

A. CSS 样式表可以将内容和外观分离

B. CSS 样式表可以控制页面的布局

C. CSS 样式表可以使许多网页同时更新

D. CSS 样式表不能制作体积更小下载更快的网页

2. CSS 样式表不可能实现(　　)功能。

A. 将内容和外观分离　　B. 一个 CSS 文件控制多个网页

C. 控制图片的精确位置　　D. 兼容所有的浏览器

3. 下面不属于 CSS 插入形式的是(　　)。

A. 索引式　　B. 内联式　　C. 嵌入式　　D. 外部链接式

4. 若要在网页中插入样式表 main.css，以下用法中，正确的是(　　)。

A. <Link　href="main.css"　type="text/css"　rel="stylesheet">

B. <Link　Src="main.css"　type="text/css"　rel="stylesheet">

C. <Link　href="main.css"　type="text/css">

D. <Include　href="main.css"　type="text/css"　rel="stylesheet">

5. 若要在当前网页中定义一个独立类的样式 myText，使具有该类样式的正文字体为"Arial"，字体大小为 9pt，行间距为 13.5pt，以下定义方法中，正确的是(　　)。

A. <Style> .myText{Font-Familiy:Arial;Font-size:9pt;Line-Height:13.5pt} </style>

B. .myText{Font-Familiy:Arial;Font-size:9pt;Line-Height:13.5pt}

C. <Style> . myText {FontName:Arial;FontSize:9pt;Line-Height:13.5pt} </style>

D. <Style> . myText {FontName:Arial;Font-size:9pt;Line-Height:13.5pt} </style>

6. 若需要动态地改变内容页的母版页，应该在页面的(　　)事件方法中进行设置。

A. Page_Load　　B. Page_Render

C. Page_PreRender　　D. Page_PreInit

7. 创建一个 Web 页面，同时也有一个名为 master.master 的母版页，要让 Web 窗体使用 master.master 母版页，应该如何处理？(　　)

A. 加入 ContentPlaceHolder 控件

B. 加入 Content 控件

C. 在 Web 页面的@Page 指令中设置 MasterPageFile 属性为 master.master，然后将窗体中<form></form>之间的内容放置在<asp:Content>…</asp:Content>内

8. 要访问母版页中的控件，可以使用(　　)。

A. 控件 ID　　B. Master.FindControl

C. Master.控件 ID　　D. 无法实现

9. 在一个 Web 站点中，有一个站点地图文件 Web.sitemap 和一个 Default.aspx 页面，在 Default.aspx 页面中包含一个 SiteMapDataSource 控件，该控件的 ID 为 smData。如果想以树状结构显示站点地图，该如何处理？(　　)

A. 拖曳一个 Menu 到页面中，并将其绑定到 SqlDataSource

B. 拖曳一个 TreeView 到页面中，并将其绑定到 SqlDataSource

C. 拖曳一个 Menu 到页面中，并设置该控件的 DataSourceID 属性设置为 smData

D. 拖曳一个 TreeView 到页面中，并设置该控件的 DataSourceID 属性设置为 smData

10. 在一个产品站点中，使用 SiteMapDataSource 控件和 TreeView 控件进行导航，站点地图 Web.sitemap 配置如下：

```
<?xml version = "1.0" encoding = "utf - 8" ?>
< siteMap xmlns = "http://schemas.microsoft.com/AspNet/SiteMap - File - 1.0" >
    < siteMapNode title = "首页" description = "网站首页" url = "~/default.aspx">
    < siteMapNode title = "产品分类" url = "~/Products.aspx" />
    < siteMapNode title = "系统管理" url = "~/Admin/Default.aspx">
    < siteMapNode title = "产品修改" url = "~/Admin/Training.aspx" />
    < siteMapNode title = "订单查询" url = "~/Admin/Consulting.aspx" />
    </siteMapNode >
    </siteMapNode >
</siteMap >
```

要求当用户进入管理员页面后，只显示管理员节点及其子节点。该如何处理？(　　)

A. 将 SiteMapDataSource 控件的 ShowStartingNode 属性设置为 false

B. 在 Admin/Default.aspx 页重新应用一个新的只包含会员节点的 Web.sitemap 地图

C. 将 SiteMapPath 控件的 SkipLinkText 属性设置为~/Admin/Default.aspx

D. 将 SiteMapDataSource 控件的 StartingNodeUrl 属性设置为~/Admin/Default.aspx

二、填空题

1. 在 ASP.NET 页面中使用 CSS 的三种方法分别是________、________、________。

2. 母版页为具有扩展名________的 ASP.NET 文件，它具有可以包括________、________和________的预定义布局。母版页由特殊@ Master 指令识别，该指令替换了用于普通.aspx 页的@Page 指令。

三、问答题

1. 简述 CSS 样式中，样式选择符可以有几种类型。
2. CSS 的主要功能是什么？
3. 简述主题中可以包含哪几类文件。
4. 阐述母版页和内容页之间的关系。
5. 简述母版页的工作原理。
6. 简述 SiteMapPath、Menu 和 TreeView 控件的用途。

单元6 数据库访问及网上书店系统架构

教学目标：

- 了解 ADO.NET。
- 熟练掌握 ADO.NET 连接 SQL 数据库。
- 熟练掌握 Connection 对象的使用。
- 熟练掌握 Command 对象的使用。
- 掌握 DataReader 对象的使用。
- 熟练掌握 DataAdapter 对象的使用。
- 掌握 DataSet 对象中常用的方法，并高效使用 DataSet 开发。
- 了解三层架构系统。
- 掌握三层架构系统的创建过程。
- 掌握每一层的代码设计。

6.1 知识准备

6.1.1 ADO.NET 概述

大部分应用程序都需要访问或者操作大量数据，通常，这些数据都是存储在数据库中的。比如到超市买东西、结账的时候，只要扫描贴在商品上的条形码，超市的结算系统就能够根据条形码从数据库中读取此商品的价格，计算出购买商品的价钱。如果没有数据库，超市出售的所有商品的价格就需要收银员输入到结算系统中，既费时又费力，还可能出现操作错误。

在信息系统中，常用的数据库有很多种，比如 SQL Server、Access、Oracle 等。为了使客户端能够访问服务器上的数据，就要用到数据库访问的方法和技术，ADO.NET(ActiveX Data Objects)就是这种技术之一。

1. ADO.NET 简介

ADO.NET 是.NET Framework 中不可缺少的一部分，它是一组类。通过这些类，.NET 应用程序就可以访问数据库。ADO.NET 的功能非常强大，它提供了对关系数据库、XML 以及其他数据存储的访问。应用程序可以通过 ADO.NET 技术与这些数据源进行连

接，对数据进行增、删、改、查等操作。

ADO.NET 技术的一个突出的优点是，它与数据源断开连接时也可以使用数据。这是因为 ADO.NET 可以把从数据源检索到的数据保存在本地内存的一个叫作 DataSet（数据集）的地方，这样应用程序就可以直接操作本地内存中的数据（也即 DataSet 中的数据），而数据源可以为更多的应用程序提供服务。这就是 ADO.NET 的断开连接模型。这就好比有一个大的工厂，工厂里有一个仓库，用来存放生产用的原料和产品。同时，工厂中有很多生产车间。假设每个车间每天要生产 100 件产品，每加工一件产品都是从仓库里取一次原料，向仓库提交一个产品，恐怕即使仓库的管理员忙得晕头转向也不能够满足所有车间的需求。所以车间就在自己的旁边建了一个临时仓库，每天先把生产用的原料一次性地从仓库中取出来放在临时仓库中，生产的时候只需要从临时仓库提取原料就行了。每天生产的产品也先寄存在临时仓库中，待下班的时候再由各车间把这一天生产的产品一次性搬运到仓库。

2. ADO.NET 的主要组件

ADO.NET 用于数据访问的类库包含 .NET Framework 数据提供程序和 DataSet（数据集）两个组件。.NET Framework 数据提供程序和 DataSet 之间的关系如图 6-1 所示。

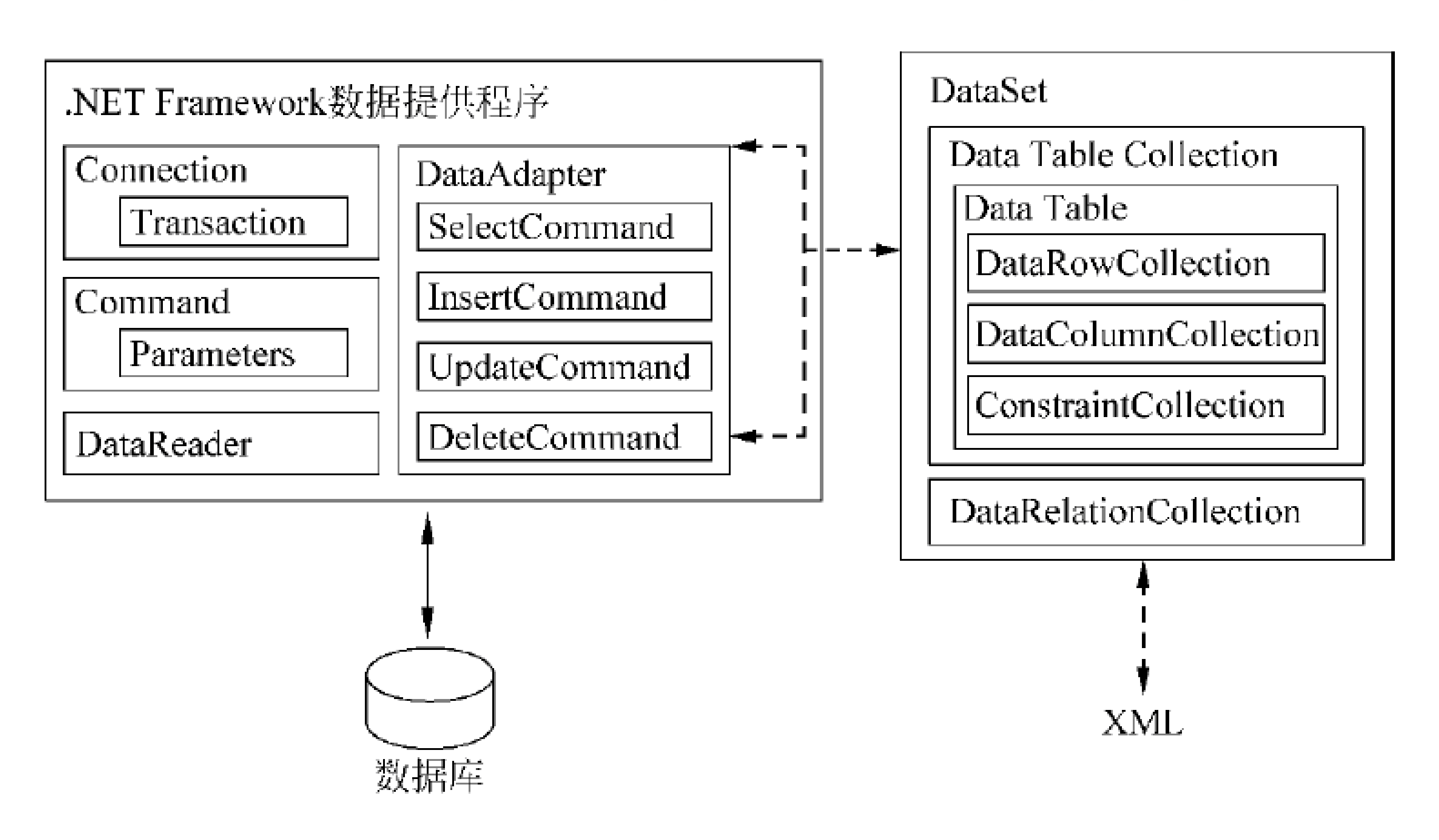


图 6-1　ADO.NET 的组成

- .NET Framework 数据提供程序是专门为数据处理以及快速地只进、只读访问数据而设计的组件。使用它可以连接数据源、执行命令和检索结果，直接对数据源进行操作。
- DataSet 是专门为独立于任何数据源的数据访问而设计的，使用它，可以不必直接和数据源打交道，可以大批量地操作数据，也可以将数据绑定在控件上。

.NET Framework 数据提供程序包含了访问各种数据源数据的对象，它是和数据库类型有关的。目前有四种类型的数据提供程序，如表 6-1 所示。

具体要使用哪种数据提供程序，要根据应用程序所使用的数据库。.NET Framework 数据提供程序包括以下四个核心对象。

- Connection：建立与数据源的连接。

- Command：对数据源执行操作命令，用于修改、查询数据和运行存储过程。
- DataReader：从数据源获取返回的数据。
- DataAdapter：用数据源数据填充 DataSet，并可以处理数据更新。

表 6-1 常见的数据提供程序及其支持的数据源描述

数据提供程序	支持数据源的描述
ODBC Data Provider	提供 ODBC 接口的数据源，包括 Access、Oracle、SQL Server、MySQL 和 Visual FoxPro 等传统数据源
OLE DB Data Provider	提供 OLE DB 接口的数据源，比如 Access、Excel、Oracle 和 SQL Server
Oracle Data Provider	用于 Oracle 数据库
SQL Data Provider	用于 Microsoft SQL Server 7 或更高版本、SQL Express 或 MSDE
Borland Data Provider	许多数据库的公共存取方式，比如 Interbase、SQL Server、IBM DB2 和 Oracle

不同的命名控件中都有相应的对象。比如要操作 SQL Server 数据库，需要使用 Systen. Data. SqlClient 命名空间，SQL 数据提供程序中的类都以 Sql 开头，所以它的四个核心对象分别为 SqlConnection、SqlCommand、SqlDataReader、SqlDataAdapter。本课程是利用 SQL Server 的 .NET 数据提供程序来操作数据库的。ADO.NET 的数据库命名空间及其说明如表 6-2 所示。

表 6-2 ADO.NET 的数据库命名空间及其说明

命名空间	说明
System. Data	ADO.NET 的核心，包含处理非连接的架构所设计的类，如 DataSet
System. Data. SqlClient	SQL Server 的 .NET 数据提供程序
System. Data. OracleClient	Oracle 的 .NET 数据提供程序
System. Data. OleDb	OLE DB 的 .NET 数据提供程序
System. Data. Odbc	ODBC 的 .NET 数据提供程序
System. Xml	提供基于标准 XML 的类、结构等
System. Data. Common	由 .NET 数据提供程序继承或者实现的工具类和接口

DataSet 是 ADO.NET 的断开式结构的核心组件。设计 DataSet 的目的是为了实现独立于任何数据源的数据访问。可以把它看成是内存中的数据库，是专门用来处理数据源中读出的数据的。

DataSet 的优点就是离线式，一旦读到数据库中的数据后，就在内存中建立数据库的副本，在此之后的操作，直到执行更新命令为止，所有的操作都是在内存中完成的。不管底层的数据库是哪种类型，DataSet 的行为都是一致的。

DataSet 是数据表(DataTable)的集合，它可以包含任意多个数据表，而且每个 DataSet 中的数据表对应一个物理数据库中的数据表(Table)或者数据视图(View)。

ASP.NET 数据访问程序的开发流程有以下几个步骤：

(1) 利用 Connection 对象创建数据连接。

(2) 利用 Command 对象对数据源执行 SQL 命令。

(3) 利用 DataReader 对象读取数据源的数据。

(4) DataSet 对象与 DataAdapter 对象配合，完成数据的查询和更新操作。

6.1.2　使用 Connection 连接数据库

当应用程序要访问数据的时候，怎样能够找到数据库呢？这就需要 Connection 对象。在 ADO.NET 对象模型中，Connection 对象用于连接数据库和管理数据库的事务。不同的数据源（.NET 数据提供程序）需要使用不同的类来建立连接。例如，要连接到 SQL Server，需要选择 SqlConnection 连接类。根据不同的数据源提供了如表 6-3 所示的 4 种数据库连接方式。

表 6-3　.NET 数据提供程序及相应的连接类

数据访问提供程序	名称空间	对应的连接类名称
SQL Server 数据提供程序	System. Data. SqlClient	SqlConnection
OLE DB 数据提供程序	System. Data. OleDb	OledbConnection
ODBC 数据提供程序	System. Data. Odbc	OdbcConnection
Oracle 数据提供程序	System. Data. OracleClient	OracleConnection

下面以 SqlConnection 为例介绍 Connection 对象的使用，其他连接方式与之类似。SqlConnection 对象提供了一些属性和方法，允许程序员与数据源建立连接或断开连接。SqlConnection 对象的常用属性和方法如表 6-4 所示。

表 6-4　SqlConnection 对象的常用属性和方法

属性或方法名称	说　　明
ConnectionString 属性	取得和设置连接字符串
ConnectionTimeOut 属性	获取 SqlConnection 对象的超时时间，单位为秒，0 表示不限制。若在这个时间之内无法连接数据源，则产生异常
Database 属性	获取当前数据库名称
DataSource 属性	获取数据源的完整路径和文件名，若是 SQL Server 数据库，则获取所连接的 SQL Server 服务器名称
State 属性	获取数据库的连接状态，它的值为 ConnectionState 枚举值
Open 方法	打开与数据库的连接
Close 方法	关闭与数据库的连接
ChangeDatabase 方法	在打开连接的状态下，更改当前数据库
CreateCommand 方法	创建并返回与 SqlConnection 对象有关的 SqlCommand 对象
Dispose 方法	调用 Close()方法关闭与数据库的连接，并释放所占用的系统资源

注意：除了 ConnectionString 之外，其他属性都是只读属性，只能通过连接字符串的标记配置数据库连接。

在 ADO.NET 中，如果使用.NET Framework 数据提供程序操作数据库，必须显式地关闭与数据库的连接，也就是说，在操作完数据库后，必须调用 Connection 对象的 Close()方法关闭连接。

建立应用程序与数据库连接需要以下三个步骤。

1. 定义连接字符串

为了连接到数据库，需要一个连接字符串。连接字符串通常由用分号隔开的名称和值

组成,不同的数据库连接字符串,其格式不同。SQL Server 数据库的连接字符串格式一般为:

```
Server=服务器名;Database=数据库名;uid=用户名;pwd=密码
```

或

```
Data Source=服务器名;Initial Catalog=数据库名; User ID=用户名;Pwd=密码
```

数据库连接字符串由以多个分号隔开的多个参数组成,其常用参数及其说明如表 6-5 所示。

表 6-5 Sqlconnection 对象的连接字符串参数及其说明

参　数	说　明
Data Source 或 Server	连接打开时使用的 SQL Server 数据库服务器名称,或者是 Microsoft Access 数据库的文件名,可以是"local"、"."、"localhost"、"127.0.0.1",也可以是具体数据库服务器的名称
Initial Catalog 或 Database	数据库的名称
Integrated Security	此参数决定连接是否是安全连接。可能的值有 True、False 和 SSPI(SSPI 是 True 的同义词)
User ID 或 uid	SQL Server 账户的登录账户
Password 或 pwd	SQL Server 登录密码

例如,"新知书店"应用程序与本机的 BookShopPlus 数据库连接的字符串可以写成:

```
String connStr="Server=.; Database=BookShopPlus; Uid=sa; pwd=123456";
```

说明:如果数据库的密码为空,可以省略 pwd 这一项。

2. 创建 Connection 对象

使用定义好的连接字符串创建 Connection 对象,代码如下。

```
SqlConnection sqlconn = new SqlConnection(connStr);
```

3. 打开与数据库的连接

调用 Connection 对象的 Open()方法打开数据库连接,代码如下。

```
sqlconn.Open();
```

在上面的这 3 个步骤中,第 1、2 步的先后顺序可以调换,即可以先创建一个 Connection 对象,再设置它的 ConnectionString 属性,如:

```
SqlConnection sqlconn = new SqlConnection();
String connStr="Server=.; Database=BookShopPlus; Uid=sa; pwd=123456";
sqlconn.ConnectionString=connStr;
```

注意：打开数据库连接，执行命令后，要确保关闭数据库连接。

【示例 6-1】 演示建立数据库连接。

(1) 在 SQL Server 2005 中附加"新知书店"项目的数据库文件 BookShopPlus.mdf。

(2) 打开 web.config 配置文件，将<connectionStrings/>标记用下面的代码替换：

```
<connectionStrings>
    <add name="connStr" connectionString="Server=.; Database=BookShopPlus;
        Uid=sa; pwd=123456"/>
</connectionStrings>
```

(3) 在 ch06 网站项目中创建文件 ConnectionDemo.aspx。

(4) 在 ConnectionDemo.aspx.cs 文件中添加相应命名空间并在 Page_Load 方法内添加如表 6-6 所示的代码。

表 6-6　页面 ConnectionDemo.aspx 的 Page_Load 事件过程的代码

```
...
using System.Configuration;     //提供对客户端应用程序配置文件
using System.Data.SqlClient;    //连接数据库
using System.Data;
...
protected void Page_Load(object sender, EventArgs e)
{
    //从 web.config 配置文件取出数据库连接字符串
    string sqlconnstr = ConfigurationManager.ConnectionStrings["connStr"].ConnectionString;
    //string sqlconnstr = "Data Source=localhost; Initial Catalog=BookShopPlus;
                            Integrated Security=True;User ID=sa; Password=";//连接字符串
    SqlConnection sqlconn = new SqlConnection(sqlconnstr);  //建立数据库连接对象
    //sqlconn.ConnectionString = "Server=localhost; Database=BookShopPlus; Uid=sa; pwd=123456";
    sqlconn.Open();             //打开连接
    if (sqlconn.State == ConnectionState.Open)
    {
        Response.Write("打开数据库连接成功!成功建立与 BookShopPlus 数据库连接");
    }
    else
    {
        Response.Write("连接数据库好像出现了意外哦!");
    }
    sqlconn.Close();            //关闭连接
    sqlconn = null;
}
```

(5) 浏览该页面，结果如图 6-2 所示。

在访问数据库之前，需要使用 Connection 对象的 Open()方法打开数据库，并在完成数据库的操作之后使用 Connection 对象的 Close()方法将数据库关闭。



图 6-2　ConnectionDemo. aspx 运行效果

6.1.3　使用 Command 对象执行数据库命令

连接数据源成功后，可以使用 Command 对象的数据库命令直接与数据源进行通信。这些命令常常包含数据库查询(select)、更新已有数据(update)、插入新数据(insert)和删除数据(delete)。许多数据库都使用结构化查询语言(SQL)来管理这些命令。Command 对象还可以调用存储过程或从特定表中取得记录。同 Connection 对象一样，Command 对象属于.NET Framework 数据提供程序，不同的数据提供程序(数据源)有各自的 Command 对象，如表 6-7 所示。

表 6-7　.NET 数据提供程序及相应的命令类

数据访问提供程序	名 称 空 间	对应的命令类名称
SQL Server 数据提供程序	System. Data. SqlClient	SqlCommand
OLE DB 数据提供程序	System. Data. OleDb	OledbCommand
ODBC 数据提供程序	System. Data. Odbc	OdbcCommand
Oracle 数据提供程序	System. Data. OracleClient	OracleCommand

在建立了数据连接之后，就可以使用相应的 Command 对象来执行数据库的操作。下面以 SqlCommand 为例进行介绍，其他与之类似。创建 Command 对象的语法如下：

```
SqlCommand command = new SqlCommand(SQL 语句, connection 对象);
```

注意：除了上述建立 SqlCommand 对象的方法外，还有 3 种：

SqlCommand 命令对象名= new SqlCommand()；

SqlCommand 命令对象名= new SqlCommand(SQL 语句)；

SqlCommand 命令对象名= new SqlCommand(SQL 语句，Connection 对象，事务对象)；

创建一个 Command 对象需要两个参数。第一个参数是将要执行的 SQL 语句，第二个参数是已经创建的 Conneciton 对象。SqlCommand 对象的主要属性和方法如表 6-8 所示。

要使用 Command 对象，必须有一个 Connection 对象。使用 Command 对象的步骤包括以下几步。

表 6-8 SqlCommand 对象的主要属性和方法

属性和方法	说 明
CommandText 属性	获取或设置要对数据源执行的 SQL 命令、存储过程或数据表名称
CommandType 属性	获取或设置命令类型，可取的值为 CommandType. Text、CommandType. StoredProduce，分别对应 SQL 命令和存储过程，默认为 Text
Connection 属性	获取或设置 SqlCommand 对象所使用的数据连接属性
Parameters 属性	SQL 命令参数集合
Cancel 方法	取消 SqlCommand 对象的执行
CreateParameter 方法	创建 Parameter 对象
ExecuteNoQuery 方法	执行 CommandText 属性指定的内容，返回数据表被影响的行数。该方法只能执行 Insert、Update 和 Delete 命令
ExecuteReader 方法	执行 CommandText 属性指定的内容，返回 DataReader 对象。该方法用于执行返回多条记录的 Select 命令
ExecuteScalar 方法	执行 CommandText 属性指定的内容，以 object 类型返回结果表第一行、第一列的值。该方法一般用来执行查询单值的 Select 命令

（1）创建数据库连接：按照前面讲过的步骤创建一个 Connection 对象。

（2）定义执行的 SQL 语句：将对数据库执行的 SQL 语句赋给一个字符串。

（3）创建 Command 对象：使用已有的 Connection 对象和 SQL 语句字符串创建一个 Command 对象。

（4）执行 SQL 语句：使用 Command 对象的某个方法执行命令。

1. 使用 Command 对象查询数据库的数据

使用 Command 对象查询数据库数据的一般步骤为：首先建立数据库连接；然后创建 Command 对象，并设置它的 Connection 和 CommandText 属性，分别表示数据库连接对象和要执行的 SQL 语句；接下来使用 Command 对象的 ExecuteReader 方法，把返回结果放在 DataReader 对象中；最后，通过循环处理查询结果。

【示例 6-2】 演示使用 Command 对象查询数据库的数据。

（1）在网站项目 ch06 中添加一个名为 CommSelectDemo. aspx 的页面。

（2）附加数据库 Student. mdf 到 SQL Server 2005 中，并添加一些记录到表 StuInfo 中，作为测试代码使用。

（3）在 CommSelectDemo. aspx. cs 文件的 Page_Load 方法内添加如表 6-9 所示的代码。

表 6-9 页面 CommSelectDemo. aspx 的 Page_Load 事件过程的代码

行号 代码页

```
protected void Page_Load(object sender, EventArgs e)
{
    string sqlconnstr = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    SqlConnection sqlconn = new SqlConnection(sqlconnstr);  //创建 Connection 对象
    SqlCommand comm = new SqlCommand();          //建立 Command 对象
```

续表

行号　代码页

```
        comm.Connection = sqlconn;                    //给 sqlcommand 的 Connection 属性赋值
        comm.CommandText = "select top 3 * from StuInfo"; //SQL 命令赋值
        try
        {
            sqlconn.Open();   //打开连接
            SqlDataReader sdr = comm.ExecuteReader();//建立 DataReader 对象,并返回查询结果
            while (sdr.Read())  //逐行遍历查询结果
            {
                Response.Write(sdr.GetString(0) + " ");
                Response.Write(sdr.GetString(1) + " ");
                Response.Write(sdr.GetString(2) + " ");
                Response.Write(sdr.GetDateTime(3).ToShortDateString() + " ");
                Response.Write(sdr.GetString(4) + "<br />");
            };
        }
        catch (Exception ex)
        {
            Response.Write("数据查询失败,原因:" + ex.Message);
        }
        finally
        {
            comm = null;
            sqlconn.Close();
            sqlconn = null;
        }
    }
```

（4）程序运行效果如图 6-3 所示。

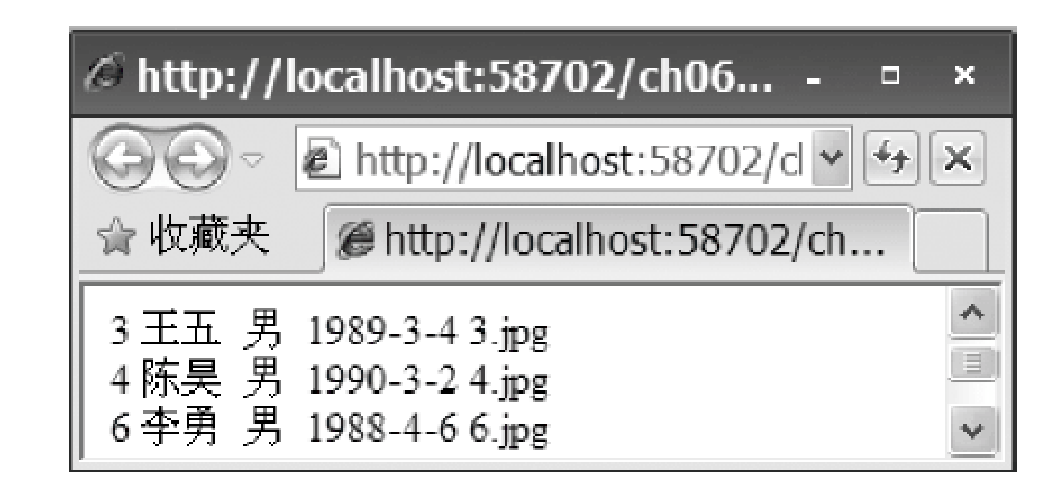


图 6-3　CommSelectDemo.aspx 运行效果

2. 使用 Command 对象增加数据库的数据

使用 Command 对象向数据库增加数据的一般步骤是：首先建立数据库连接；然后创建 Command 对象，并设置它的 Connection 和 CommandText 属性，即使用 Command 对象的 Parameters 属性来设置输入参数；最后，使用 Command 对象的 ExecuteNoQuery 方法执

行数据库增加命令,ExecuteNoQuery 方法表示要执行的是没有返回数据的命令。

【示例 6-3】 演示使用 Command 对象向数据库中添加新数据。

(1) 在网站项目 ch06 中,新建一个名为 images 的文件夹,用于存放学生照片。

(2) 在网站项目 ch06 中添加一个名为 CommInsertDemo.aspx 的页面。

(3) 在 CommInsertDemo.aspx 中添加 Web 控件,使其外观如图 6-4 所示,代码如表 6-10 所示。

表 6-10　页面 CommInsertDemo.aspx 的 XHTML 代码

```
<form id = "form1" runat = "server">
<div>
    <table style = "width: 320px; height: 240px">
        <tr>
            <td style = "width: 100px; text - align: right"> 学号: </td>
            <td style = "width: 220px">
                <asp:TextBox ID = "txtStuNo" runat = "server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td style = "width: 100px; text - align: right">姓名: </td>
            <td style = "width: 220px">
                <asp:TextBox ID = "txtName" runat = "server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td style = "width: 100px; text - align: right">性别: </td>
            <td style = "width: 220px">
                <asp:DropDownList ID = "DdSex" runat = "server">
                    <asp:ListItem Selected = "True">男</asp:ListItem>
                    <asp:ListItem>女</asp:ListItem>
                </asp:DropDownList>
            </td>
        </tr>
        <tr>
            <td style = "width: 100px; text - align: right">出生日期: </td>
            <td style = "width: 220px">
                <asp:TextBox ID = "txtBirth" runat = "server"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td style = "width: 100px; text - align: right">照片: </td>
            <td style = " width: 220px" > < asp: FileUpload ID = " FileUpload1" runat = " server" /></td>
        </tr>
```

续表

行号　代　码　页

```
        <tr>
            <td colspan="2" style="text-align: center">
                <asp:Button ID="btnAdd" runat="server" Text="提交" OnClick="btnAdd_Click" />
            </td>
        </tr>
    </table>
</div>
</form>
```

（4）编写 CommInsertDemo.aspx 页面中“提交”按钮的 Click 事件方法的程序代码，如表 6-11 所示。

表 6-11　“提交”按钮的 Click 事件方法的代码

行号　代　码　页

```
protected void btnAdd_Click(object sender, EventArgs e)
{
     string sqlconnstr = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    SqlConnection sqlconn = new SqlConnection(sqlconnstr); //根据 sqlconnstr 创建 connection 对象
    SqlCommand cmd = new SqlCommand();  //建立 Command 对象
    cmd.Connection = sqlconn;                //指定 Command 对象的 connection 属性值
    cmd.CommandText = "insert into StuInfo(StuNo,Name,Sex,Birth,Photo)values
                (@StuNo,@Name,@Sex,@Birth,@Photo)"; //把 SQL 语句赋给 Command 对象
    //在执行之前告诉 Command 对象@StuNo、@Name、@Sex、@Birth、@Photo 将来用谁来代替
    SqlParameter[] paras = new SqlParameter[] {
                new SqlParameter("@StuNo", txtStuNo.Text),
                new SqlParameter("@Name", txtName.Text),
                new SqlParameter("@Sex", DdSex.Text),
                new SqlParameter("@Birth", txtBirth.Text),
                new SqlParameter("@Photo", txtStuNo.Text + ".jpg")
                };
    cmd.Parameters.AddRange(paras); //给 Command 对象添加参数,以数组形式添加
    try
    {
    sqlconn.Open();          //打开连接
        cmd.ExecuteNoQuery(); //执行 SQL 命令
        //把学生的照片上传到网站的"images"文件夹中,以学号为文件名进行保存
        if (FileUpload1.HasFile == true)
        {
            string fileName = this.txtStuNo.Text + ".jpg";
```

续表

```
            FileUpload1.SaveAs(Server.MapPath(("~/images/") + fileName));
        }
        Response.Write("成功追加记录");
    }
    catch (Exception ex)
    {
        Response.Write("错误原因: " + ex.Message);
    }
    finally
    {
        cmd = null;
        sqlconn.Close();
        sqlconn = null;
    }
}
```

（5）程序运行效果如图 6-4 所示。





图 6-4　CommInsertDemo. aspx 页面设计及运行效果

3. 使用 Command 对象删除数据库的数据

使用 Command 对象向数据库增加数据的一般步骤是：首先建立数据库连接；然后创建 Command 对象，并设置它的 Connection 和 CommandText 属性，即使用 Command 对象的 Parameters 属性来设置输入参数；最后，使用 Command 对象的 ExecuteNoQuery 方法执行数据删除命令。

【示例 6-4】　演示使用 Command 对象删除数据。

（1）在网站项目 ch06 中添加一个名为 CommDeleteDemo. aspx 的页面。

(2) 在 CommDeleteDemo.aspx 中添加一个 TextBox 控件和一个 Button 控件，其中 Button 控件作为“删除”按钮，代码如下。

```
<form id="form1" runat="server">
<div>
    输入要删除学生的学号：<br />
   <asp:TextBox ID="txtStuNo" runat="server"></asp:TextBox>
   <asp:Button ID="btnDel" runat="server" Text="删除" OnClick="btnDel_Click" />
</div>
</form>
```

(3) 编写 CommDeleteDemo.aspx 页面中“删除”按钮的 Click 事件方法的程序代码，如表 6-12 所示。

表 6-12 “删除”按钮的 Click 事件方法的代码

行号　　代　码　页

```
protected void btnDel_Click(object sender, EventArgs e)
{
    int intDeleteCount;
    string sqlconnstr = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    SqlConnection sqlconn = new SqlConnection(sqlconnstr);
    SqlCommand cmd = new SqlCommand();   //建立 Command 对象
    //下两行分别给 Command 对象的 Connection 和 CommandText 属性赋值
    cmd.Connection = sqlconn;
    cmd.CommandText = "delete from StuInfo where StuNo=@no";
    //在执行之前告诉 Command 对象@no 将来用谁来代替,注意与示例 6-3 的区别
    SqlParameter p1 = new SqlParameter("@No", txtStuNo.Text);
    cmd.Parameters.Add(p1);                //添加 Command 命令参数
    try
    {
        sqlconn.Open();
        intDeleteCount = cmd.ExecuteNoQuery();
        if (intDeleteCount > 0)
            Response.Write("删除成功!");
        else
            Response.Write("该记录不存在!");
    }
    catch (Exception ex)
    {
        Response.Write("删除失败,错误原因：" + ex.Message);
    }
    finally
    {
        cmd = null;
        sqlconn.Close();
```

续表

```
            sqlconn = null;
        }
    }
```

（4）程序运行效果如图 6-5 所示。

4. 使用 Command 对象修改数据库的数据

使用 Command 对象修改数据库数据的一般步骤是：首先建立数据库连接；然后创建 Command 对象，并设置它的 Connection 和 CommandText 属性，即使用 Command 对象的 Parameters 属性来设置输入参数；最后，使用 Command 对象的 ExecuteNoQuery 方法执行数据库修改命令。

修改数据库和往数据库中添加数据的操作类似，在此不再举例，请读者自行完成。

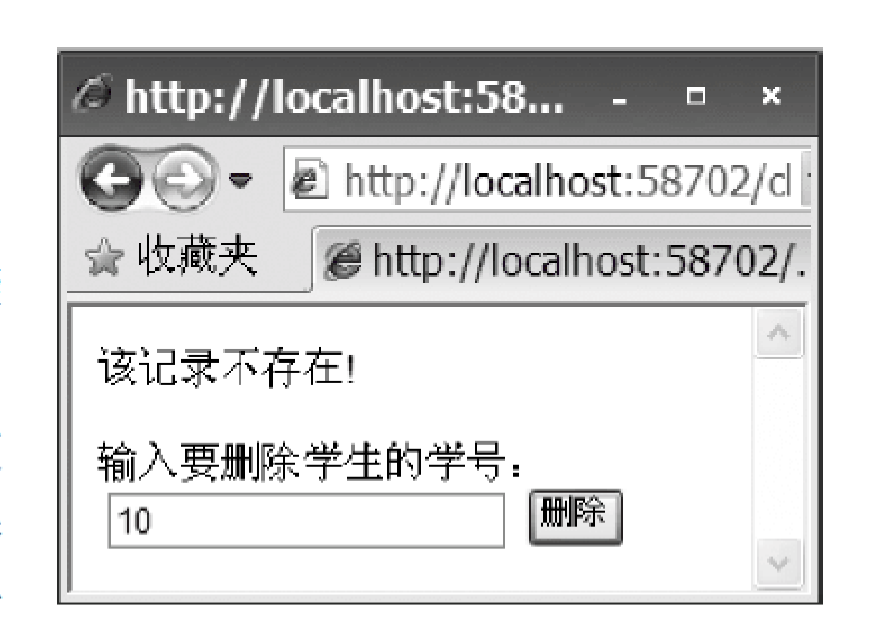


图 6-5　页面 CommDeleteDemo.aspx 运行效果

6.1.4　使用 DataReader 对象执行数据库命令

1. DataReader 对象概述

ADO.NET 的 DataReader 对象可以从数据源中检索只读、只进的数据集，用于快速读取数据。对于只需要顺序显示数据表中记录的应用而言，DataReader 对象是比较理想的选择。在读取数据时，它需要与数据源保持实时连接，以循环的方式读取结果集中的数据。

DataReader 对象不能直接实例化，而必须调用 Command 对象的 ExecuteReader 方法才能创建有效的 DataReader 对象。DataReader 对象一旦创建，即可通过对象的属性、方法访问数据源中的数据。DataReader 属于.NET 数据提供程序，每一种.NET 数据提供程序都有与之对应的 DataReader 类，如表 6-13 所示。

表 6-13　.NET 数据提供程序及相应的 DataReader 类

数据访问提供程序	名 称 空 间	对应的 DataReader 类名称
SQL Server 数据提供程序	System.Data.SqlClient	SqlDataReader
OLE DB 数据提供程序	System.Data.OleDb	OledbDataReader
ODBC 数据提供程序	System.Data.Odbc	OdbcDataReader
Oracle 数据提供程序	System.Data.OracleClient	OracleDataReader

下面以 SqlDataReader 为例进行介绍，其他与之类似。SqlDataReader 对象的主要属性和方法如表 6-14 所示。

2. DataReader 对象的用法

创建一个 DataReader 对象需要调用 Command 对象的 ExecuteReader 方法，

ExecuteReader 的返回值是一个 DataReader 对象。可以调用 DataReader 对象的 Read()方法读取一行记录。

表 6-14　SqlDataReader 对象的主要属性和方法

属性和方法	说　明
FieldCount 属性	获取由 DataReader 得到的一行数据中的字段数
isClosed 属性	获取 SqlDataReader 对象的状态。True 表示关闭，False 表示打开
HasRows 属性	表示查询是否返回结果。如果返回查询结果则返回 True，否则返回 False
HasMoreRows 属性	只读，表示是否还有记录未读取
Close 方法	不带参数，无返回值，用来关闭 DataReader 对象
Read 方法	让记录指针指向本结果集中的下一条记录，返回值是 True 或 False
NextResult 方法	当返回多个结果集时，使用该方法让记录指针指向下一个结果集。当调用该方法获得下一个结果集后，依然要用 Read 方法来遍历访问该结果集
GetValue 方法	根据传入的列的索引值，返回当前记录行里指定列的值。由于事先无法预知返回列的数据类型，所以该方法使用 Object 类型来接收返回数据
GetValues 方法	该方法会把当前记录行里所有的数据保存到一个数组里。可以使用 FieldCount 属性来获知记录里字段的总数，据此定义接收返回值的数组长度
GetName 方法	通过输入列索引获得该列的名称。综合使用 GetName 和 GetValue 方法，可以获得数据表里列名和列的字段
IsDBNull 方法	判断指定索引号的列的值是否为空，返回 True 或 False

创建和使用 SqlDataReader 的步骤如下。

(1) 创建 SqlConnection 对象，设置连接字符串。

(2) 创建 SqlCommand 对象，设置它的 Connection 和 CommandText 属性，分别表示数据库连接和需要执行的 SQL 命令。

(3) 打开与数据库连接。

(4) 使用 SqlCommand 对象的 ExecuteReader 方法执行 CommandText 中的命令，并把返回的结果放在 SqlDataReader 对象中。假设已创建一个名为 cmd 的 Command 对象，下面的代码可以创建一个 DataReader 对象。

```
SqlDataReader  dr = cmd.ExecuteReader();
```

(5) 通过调用 SqlDataReade 对象的 Read()方法循环读取查询结果集的记录。这个方法返回一个布尔值。如果能读到一行记录，则返回 True，否则返回 False。代码如下。

```
Dr.Read();
```

(6) 读取当前行的某列的数据。可以像使用数组一样，用方括号来读取某列的值，如(type)dr[ ]，方括号中可以是列的索引，(从 0 开始)也可以是列名。对取到的列值必须要进行类型转换，如下所示。

```
(string)dr["name"];
```

(7) 关闭与数据库连接。

【示例 6-5】 演示使用 SqlDataReade 对象读取数据库中的数据。

（1）在网站项目 ch06 中添加一个名为 DataReaderDemo.aspx 的页面。

（2）编写 DataReaderDemo.aspx 页面 Page_Load 事件方法的程序代码，如表 6-15 所示。

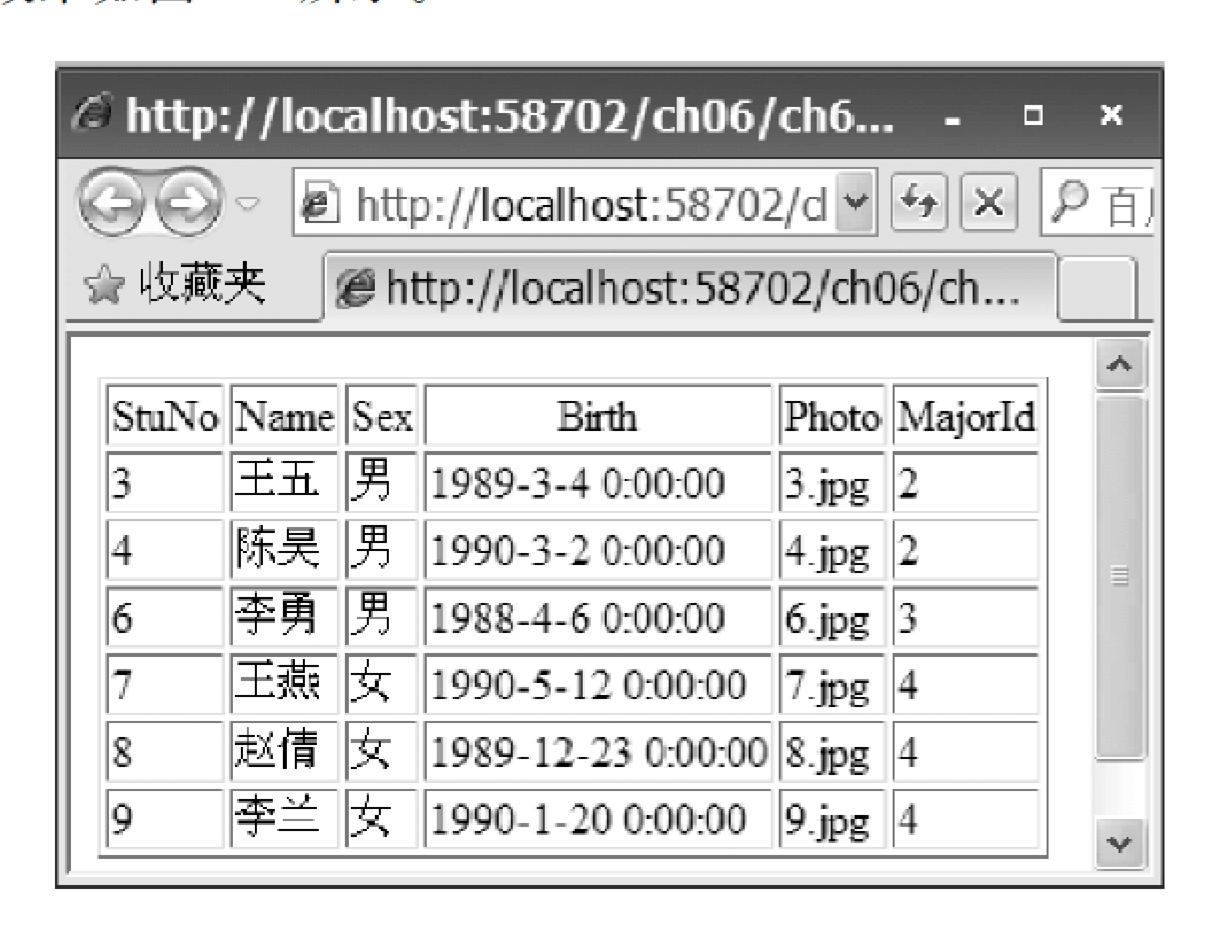
表 6-15　页面 DataReaderDemo.aspx 的 Page_Load 事件方法的代码

```
protected void Page_Load(object sender, EventArgs e)
{
    string strCnn = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    SqlConnection cnn = new SqlConnection(strCnn);
    SqlCommand cmd = new SqlCommand();
    cmd.Connection = cnn;
    cmd.CommandText = "select * from StuInfo";
    SqlDataReader dr = null;  //创建 DataReader 对象的引用
    try
    {
        if (cnn.State == ConnectionState.Closed)
        cnn.Open();
        dr = cmd.ExecuteReader();  //执行 SQL 命令,并获取查询结果
        //依次读取查询结果的字段名称,并以表格的形式显示
        Response.Write("<table border='1'><tr align='center'>");
        for (int i = 0; i < dr.FieldCount; i++)
        {
            Response.Write("<td>" + dr.GetName(i) + "</td>");
        }
        Response.Write("</tr>");
        while (dr.Read())  //如果 DataRead 对象成功获得数据,返回 True,否则返回 False
        {
            Response.Write("<tr>");
            for (int j = 0; j < dr.FieldCount; j++)  //依次读取查询结果的字段值,并以
                                                     //表格的形式显示
            {
                Response.Write("<td>" + dr.GetValue(j) + "</td>");
            }
            Response.Write("</tr>");
        }
        Response.Write("</table>");
    }
    catch (Exception ex)
    {
        Response.Write("SqlDataReader 读取出错,原因: " + ex.Message);
    }
    finally
```

续表

```
        {
            if (dr.IsClosed == false)
                dr.Close();          //关闭 DataReader 对象
            if (cnn.State == ConnectionState.Open)
                cnn.Close();
        }
    }
```

（3）程序运行效果如图 6-6 所示。



StuNo	Name	Sex	Birth	Photo	MajorId
3	王五	男	1989-3-4 0:00:00	3.jpg	2
4	陈昊	男	1990-3-2 0:00:00	4.jpg	2
6	李勇	男	1988-4-6 0:00:00	6.jpg	3
7	王燕	女	1990-5-12 0:00:00	7.jpg	4
8	赵倩	女	1989-12-23 0:00:00	8.jpg	4
9	李兰	女	1990-1-20 0:00:00	9.jpg	4

图 6-6　页面 DataReaderDemo.aspx 运行效果

使用 SqlDataReader 对象时，应注意以下几点：

- 读取数据时，SqlConnection 对象必须处于打开状态。
- 必须通过调用 SqlCommand 对象的 ExecuteReader()方法产生 SqlDataReader 对象的实例。
- 只能按向下的顺序逐条读取记录，不能随机读取，且无法直接获知读取记录的总数。
- SqlDataReader 对象管理的查询结果是只读的，不能修改。

6.1.5　使用 DataSet 和 DataAdapter 对象

1. DataSet 对象

DataSet 是 ADO.NET 的核心组件之一，位于 System.Data 命名空间下。可以简单地把 DataSet 数据集理解为一个临时的内存数据库，可以存放多个表(DataTable)，而且是断开式的。DataSet 为数据源提供了一个断开式的存储空间，即从数据库完成数据抽取后，DataSet 就是数据的存放地，它是各种数据源中的数据在计算机内存中映射成的缓存，当应用程序需要数据时，就直接从内存中的 DataSet 数据集读取数据，也可以对 DataSet 数据集中的数据进行修改，然后将修改后的数据一起提交给数据库。

1）DataSet 对象结构模型

DataSet 数据集对象的结构和我们熟悉的 SQL Server 非常相似，如图 6-7 所示。在 SQL Server 数据库中，有很多数据表，每个数据表都有行和列。DataSet 数据集中也包含多个表，这些表构成了一个数据表集合(DataTableColleciton)，其中每个数据表都是一个 DataTable 对象。在每个数据表中又有列和行，所有的列构成了数据列集合(DataColumnCollection)，而每个数据列是一个 DataColumn 对象。所有的行构成了数据行集合(DataRowCollection)，每一行是一个 DataRow 对象。此外，在 DataSet 中还可以定义表之间的链接、视图等。

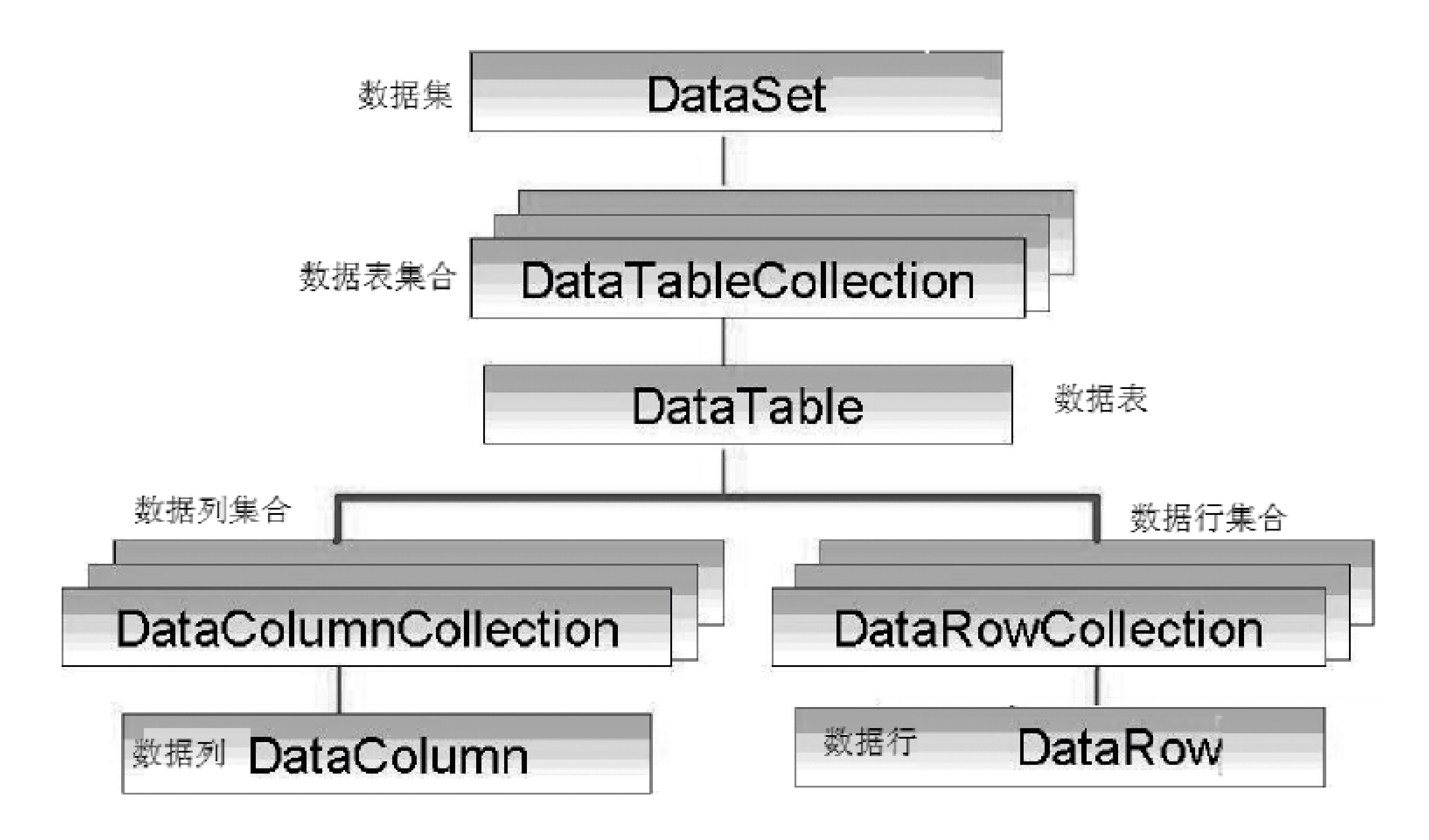


图 6-7　DataSet 数据集对象基本结构

注意：各个数据表 DataTable 之间的关系通过 DataRelation 来表示，这些 DataRelation 构成的集合就是 DataRelationColleciton 对象，如图 6-1 所示。

2）DataSet 对象工作原理

DataSet 数据集对象工作原理如图 6-8 所示。

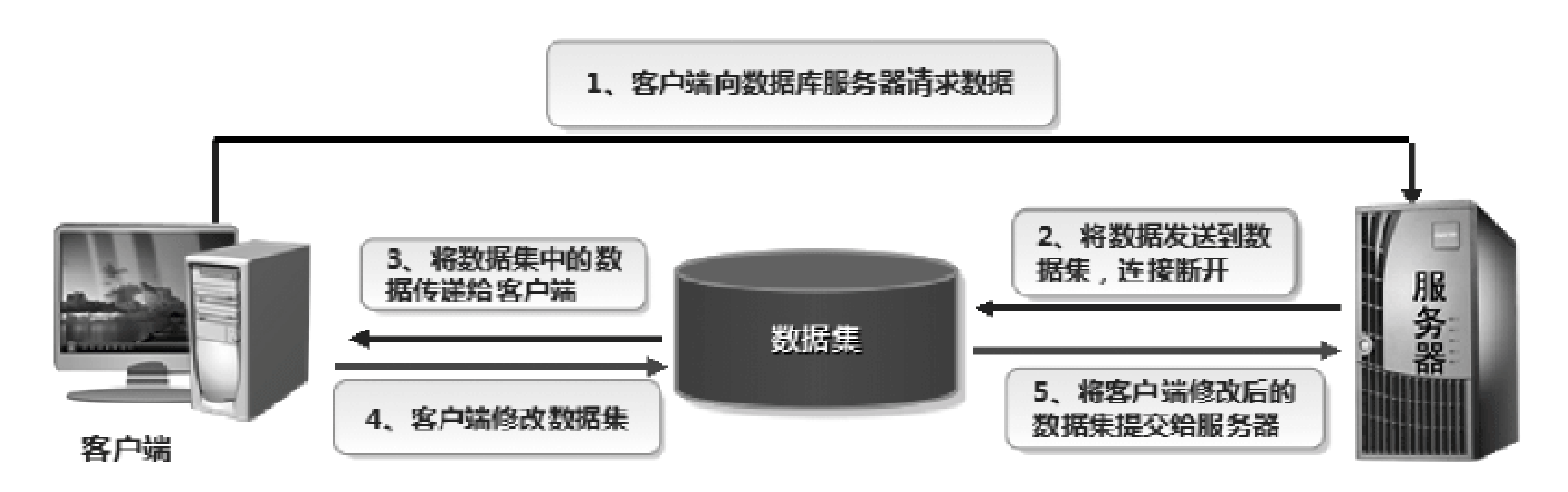


图 6-8　DataSet 数据集对象工作原理

当应用程序需要获取一些数据的时候，先向数据库服务器发出请求，要求获得数据。服务器将数据发送到 DataSet 数据集，然后再将数据集传递给客户端。客户端应用程序修改数据集中的数据后，统一将修改过的数据集发送到服务器，服务器接收数据集并修改数据库中的数据。

3）DataSet(数据集)对象创建

创建 DataSet 的语法格式为：

```
DataSet 对象名 = new DataSet();
```

或

```
DataSet 对象名 = new DataSet("数据集名");
```

例如，创建数据集对象 dsStu，代码如下：

```
DataSet dsStu = new DataSet();
```

或

```
DataSet dsStu = new DataSet("Student");
```

注意：方法中的参数是数据集的名称字符串，可以有，也可以没有。如果没有写参数，创建的数据集名称就默认为 NewDataSet。

DataSet 对象的常用属性和方法如表 6-16 所示。

表 6-16 DataSet 对象的常用属性和方法

属性和方法	说　明
DataSetName 属性	获取或设置 DataSet 对象的名称
Tables 属性	获取包含在 DataSet 数据集中的数据表的集合
Clear 方法	删除 DataSet 对象中的所有表
Copy 方法	复制 DataSet 的结构和数据，返回与本 DataSet 对象具有相同结构和数据的 DataSet 对象

4) DataTable(数据表)对象的创建

DataSet 中的每个数据表都是一个 DataTable 对象。定义 DataTable 对象的语法格式为：

```
DataTable 对象名 = new DataTable();
```

或

```
DataTable 对象名 = new DataTable("数据表名");
```

例如，创建数据表对象 dtStuInfo，代码如下：

```
DataTable dtStuInfo = new DataTable();
dtStuInfo.TableName = "StuInfo";
```

或

```
DataTable dtStuInfo = new DataTable("StuInfo");
```

创建好的数据表对象，可以被添加到数据集对象中，例如把创建好的 dtStuInfo 数据表对象添加到数据集对象 dsStu 中的代码如下：

```
dsStu.Tables.Add(dtStuInfo);
```

DataTable 对象的常用属性和方法如表 6-17 所示。

表 6-17　DataTable 对象的常用属性和方法

属性和方法	说　明
Columns 属性	获取数据表的所有字段
DataSet 属性	获取 DataTable 对象所属的 DataSet 对象
DefaultView 属性	获取与数据表相关的 DataView 对象
PrimaryKey 属性	获取或设置数据表的主键
Rows 属性	获取数据表的所有行
TableName 属性	获取或设置数据表名
Clear()方法	清除表中所有的数据
NewRow()方法	创建一个与当前数据表有相同字段结构的数据行

5）DataColumn(数据列)对象的创建

DataTable 对象中包含多个数据列，每列就是一个 DataColumn 对象。定义 DataColumn 对象的语法格式为：

```
DataColumn 对象名 = new DataColumn();
```

或

```
DataColumn 对象名 = new DataColumn("字段名");
```

或

```
DataColumn 对象名 = new DataColumn("字段名",数据类型);
```

例如，创建数据列对象 stuNoColumn，代码如下：

```
DataColumn stuNoColumn = new DataColumn();
stuNoColumn.ColumnName = " StuNo";
stuNoColumn.DataType  = System.Type.GetType("System.String");
```

或

```
DataColumn stuNoColumn = new DataColumn("StuNo", System.Type.GetType("System.String"));
```

创建好的数据列对象，可以被添加到数据表对象中，例如，将创建的数据列对象 stuNoColumn 添加到刚才创建的数据表对象 dtStuInfo 中的代码如下：

```
dtStuInfo.Columns.Add(stuNoColumn);
```

DataColumn 对象的常用属性如表 6-18 所示。

表 6-18 对象的常用属性和方法

属　性	说　明
AllowDBNull	设置该字段可否为空值。默认为 true
Caption	获取或设置字段标题。若指定字段标题，则字段标题与字段名相同
ColumnName	获取或设置字段名
DataType	获取或设置字段的数据类型
DefaultValue	获取或设置新增数据行时，字段的默认值

说明：通过 DataColumn 对象的 DataType 属性设置字段数据类型时，不可直接设置数据类型，而要按照以下语法格式：

```
对象名.DataType = System.Type.GetType("数据类型"));
```

6）DataRow（数据行）的创建

DataTable 对象可以包含多个数据行，每行就是一个 DataRow 对象。定义 DataRow 对象的语法格式为：

```
DataRow 对象名 = DataTable 对象.NewRow();
```

注意：DataRow 对象不能用 New 来创建，而需要用数据表对象的 NewRow 方法创建。

例如，为数据表对象 dtStuInfo 添加一个新的数据行，代码如下：

```
DataRow dr = dtStuInfo.NewRow();
```

访问一行中某个单元格内容的方法为：

```
DataRow 对象名["字段名"]或 DataRow 对象名[序号]
```

DataRow 对象的常用属性和方法如表 6-19 所示。

表 6-19 DataRow 对象的常用属性和方法

属性和方法	说　明
RowState 属性	获取数据行的当前状态，属于 DataRowState 枚举型，分别为 Add、Delete、Detached、Modified、Unchanged
BeginEdit 方法	开始数据行的编辑
CancelEdit 方法	取消数据行的编辑
Delete 方法	删除数据行
EndEdit 方法	结束数据行的编辑

2. DataAdapter 对象

DataAdapter 是一个特殊的类，起着 Connection 对象和 DataSet 对象之间桥梁的作用，

能够保存和检索数据。DataAdapter 提供了双向的数据传输机制，它可以在数据源上执行 Select 语句，把查询结果集传送到 DataSet 对象的数据表（DataTable）中，还可以执行 Insert、Update 和 Delete 语句，将 DataTable 对象更改过的数据提取并更新回数据源。

DataAdapter 属于.NET 数据提供程序，每一种.NET 数据提供程序都有与之对应的 DataAdapter 类，如表 6-20 所示。

表 6-20　.NET 数据提供程序及相应的 DataAdapter 类

数据访问提供程序	名 称 空 间	对应的 DataAdapter 类名称
SQL Server 数据提供程序	System. Data. SqlClient	SqlDataAdapter
OLE DB 数据提供程序	System. Data. OleDb	OledbDataAdapter
ODBC 数据提供程序	System. Data. Odbc	OdbcDataAdapter
Oracle 数据提供程序	System. Data. OracleClient	OracleDataAdapter

下面以 SqlDataAdapter 为例进行介绍，其他与之类似。SqlDataAdapter 对象的主要属性如下。

- SelectCommad：获取或设置一个语句或存储过程，用于在数据库中选择记录。
- InsertCommand：获取或设置一个语句或存储过程，用于在数据库中插入记录。
- UpdateCommand：获取或设置一个语句或存储过程，用于更新数据库中的记录。
- DeleteCommand 属性：获取或设置一个语句或存储过程，用于从 DataSet 数据集中删除记录。

SqlDataAdapter 对象的主要方法如下。

- Fill 方法：调用 Fill 方法会自动执行 SelectCommand 属性中提供的命令，获取结果集并填充数据集的 DataTable 对象。其本质是通过执行 SelectCommand 对象的 Select 语句查询数据库，返回 DataReader 对象，通过 DataReader 对象隐式地创建 DataSet 中的表，并向 Dataset 中的表填充数据。
- Update 方法：调用 InsertCommand、UpdateCommand 和 DeleteCommand 属性指定的 SQL 命令，将 DataSet 对象更新到相应的数据源。在 Update 方法中，逐行检查数据表每行的 RowState 属性值，根据不同的 RowState 属性，调用不同的 Command 命令更新数据库。

使用 SqlDataAdapter 和 DataSet 对象操作数据库的步骤为：

（1）创建数据库连接对象。

（2）利用数据库连接对象和 Select 语句创建 SqlDataAdapter 对象。

（3）根据操作要求配置 SqlDataAdapter 对象中不同的 Command 属性。如增加数据库数据，需要配置 InsertCommand 属性；修改数据库数据，需要配置 UpdateCommand 属性；删除数据库数据，需要配置 DeleteCommand 属性。

（4）使用 SqlDataAdapter 对象的 Fill 方法把 Select 语句的查询结果放在 DataSet 对象的一个数据表中或直接放在一个 DataTable 对象中。

（5）对 DataTable 对象中的数据进行增、删、改操作。

（6）修改完成后，通过 SqlDataAdapter 对象的 Update 方法将 DataTable 对象中的修改更新到数据库。

说明：第(3)步中根据操作要求配置 SqlDataAdapter 对象中不同的 Command 属性，如果自己给 SqlDataAdapter 对象的 InsertCommand、UpdateCommand、DeleteCommand 属性定义 SQL 更新语句，过程比较复杂。可以通过建立 CommandBuilder 对象以便自动生成 DataAdapter 的 Command 命令。

DataAdapter 类的构造方法有多个，以 SqlDataAdapter 为例，构造方法如下。

```
public SqlDataAdapter();
public SqlDataAdapter(SqlCommand);
public SqlDataAdapter(string,SqlConnection);
public SqlDataAdapter(string,string);
```

故创建 SqlDataAdapter 对象的格式有以下 4 种方式。

方式一：

```
SqlDataAdapter() sda = new SqlDataAdapter();
sda.SelectCommand = new SqlCommand("select * from StuInfo", sqlconn);
```

其中 sqlconn 表示连接数据库对象，select * from StuInfo 表示 select 查询语句。

方式二：

```
SqlCommand cmd = new SqlCommand("select * from StuInfo", sqlconn);
SqlDataAdapter() sda = new SqlDataAdapter(cmd);
```

方式三：

```
SqlDataAdapter() sda = new SqlDataAdapter("select * from StuInfo", sqlconn);
```

方式四：

```
SqlDataAdapter() sda = new SqlDataAdapter("select * from StuInfo",
"Server = .; Database = BookShopPlus; Uid = sa; pwd = 123456");
```

以上是创建 SqlDataAdapter 对象的几种方式，读者可以根据需求，选择自己喜欢的方式。

3. DataSet 和 DataAdapter 对象应用

1）查询数据库的数据

使用 DataSet 对象和 DataAdapter 对象查询数据库数据的一般步骤如下：首先建立数据库连接；然后利用数据连接和 Select 语句建立 DataAdapter 对象，并使用 DataAdapter 对象的 Fill 方法把查询结果放在 DataSet 对象的一个数据表中；接下来，将该数据表复制到 DataTable 对象中；最后，实施对 DataTable 对象中数据的查询。

【示例 6-6】 演示使用 DataSet 对象和 DataAdapter 对象查询数据库数据。

（1）在网站项目 ch06 中添加一个名为 DataAdapterSelectDemo.aspx 的页面。

（2）编写 DataAdapterSelectDemo.aspx 页面 Page_Load 事件方法的程序代码，如表 6-21 所示。

表 6-21　DataAdapterSelectDemo. aspx 页面 Page_Load 事件方法的程序代码

```
protected void Page_Load(object sender, EventArgs e)
{
    string sqlconnstr = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    SqlConnection sqlconn = new SqlConnection(sqlconnstr);
    DataSet ds = new DataSet();  //建立 DataSet 对象
    DataRow dr;                  //建立 DataRow 数据行对象
    try
    {
        sqlconn.Open(); //打开连接
        //建立 DataAdapter 对象
        SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", sqlconn);
        sda.Fill(ds, "StuTable");//用 Fill 方法返回的数据,填充 DataSet,数据表取名为
                                 //"StuTable"
        DataTable dtable = ds.Tables["StuTable"];  //将数据表 StuTable 的数据复制到
                                                   //DataTable 对象
        //用 DataRowCollection 数据行对象获取 StuTable 数据表的所有数据行
        DataRowCollection drc = dtable.Rows;
        //逐行遍历,取出各行的数据
        for (int i = 0; i < drc.Count; i++)
        {
            dr = drc[i];
            Response.Write(" 学号: " + dr ["StuNo"] + "姓名:" + dr["Name"] + "性别:" +
                dr["Sex"] + "出生日期: " + dr["Birth"] + " 照片: " + dr["Photo"]);
            Response.Write("<br />");
        }
    }
    catch (Exception ex)
    {
        Response.Write("数据读取出错!原因: " + ex.Message);
    }
    finally
    {
        sqlconn.Close();
        sqlconn = null;
    }
}
```

(3) 程序运行效果如图 6-9 所示。

在后续章节会介绍绑定控件 GridView,可使显示 DataSet 中的数据更加简单。

2) 新增数据库的数据

使用 DataSet 对象和 DataAdapter 对象增加数据库数据的一般步骤如下：首先建立数



图 6-9　页面 DataAdapterSelectDemo.aspx 运行效果

据库连接；然后利用数据连接和 Select 语句建立 DataAdapter 对象并建立 CommandBuilder 对象以便自动生成 DataAdapter 的 Command 命令，否则，就要自己给 InsertCommand、UpdateCommand、DeleteCommand 属性定义 SQL 更新语句；使用 DataAdapter 对象的 Fill 方法把 Select 查询的结果放在 DataSet 对象的一个数据表中；接下来，将该数据表复制到 DataTable 对象中；最后，向 DataTable 对象增加数据记录，并通过 DataAdpter 对象的 Update 方法向数据库提交数据。

【示例 6-7】 演示使用 DataSet 对象和 DataAdapter 对象向数据库增加一条学生记录。

（1）在网站项目 ch06 中添加一个名为 DataAdapterInsertDemo.aspx 的页面。

（2）编写 DataAdapterInsertDemo.aspx 页面中“提交”按钮的 Click 事件方法的程序代码，如表 6-22 所示。

表 6-22　“提交”按钮的 Click 事件方法的程序代码

行号	代　码　页
01	`protected void btnAdd_Click(object sender, EventArgs e)`
02	`{`
03	`    string sqlconnstr = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;`
04	`    SqlConnection sqlconn = new SqlConnection(sqlconnstr);`
05	`    DataSet ds = new DataSet();//建立 DataSet 对象`
06	`    try`
07	`    {`
08	`        sqlconn.Open(); //打开连接`
09	`        SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", sqlconn); //建立 DataAdapter 对象`
10	`        //建立 CommandBuilder 对象来自动生成 DataAdapter 的 Command 命令,否则就要自己编写`
11	`        //Insertcommand,deletecommand, updatecommand 命令`
12	`        SqlCommandBuilder cb = new SqlCommandBuilder(sda);`
13	`        sda.Fill(ds, "stuTable");//用 Fill 方法返回的数据,填充 DataSet,数据表取名为 //"stuTable"`

续表

```
            DataTable dtable = ds.Tables["stuTable"];    //将数据表 stuTable 的数据复制到
                                                         //DataTable 对象
            if (FileUpload1.HasFile == true)//将照片上传到网站的"images"文件夹中,以学号
                                             //为名字进行保存
            {
                string fileName = this.txtStuNo.Text + ".jpg";
                FileUpload1.SaveAs(Server.MapPath(("~/images/") + fileName));
            }
            DataRow dr = ds.Tables["stuTable"].NewRow(); //增加新记录
            //给该记录赋值
            dr["StuNo"] = txtStuNo.Text.Trim();
            dr["Name"] = txtName.Text.Trim();
            dr["Sex"] = DdSex.SelectedValue;
            dr["Birth"] = Convert.ToDateTime(txtBirth.Text.Trim());
            dr["Photo"] = txtStuNo.Text.Trim() + ".jpg";
            dtable.Rows.Add(dr);
            sda.Update(ds, "stuTable");  //提交更新
            Response.Write("增加成功<hr>");
        }
        catch (Exception ex)
        {
            Response.Write("记录新增失败,原因: " + ex.Message);
        }
        finally
        {
            sqlconn.Close();
            sqlconn = null;
            Response.Write("<h7>成功关闭 SQL Server 数据库的连接</h7><hr>");
        }
    }
```

(3) 程序运行效果如图 6-10 所示。

3) 修改数据库的数据

使用 DataSet 对象和 DataAdapter 对象修改数据库数据的一般步骤如下：首先建立数据库连接；然后利用数据连接和 Select 语句建立 DataAdapter 对象并建立 CommandBuilder 对象以便自动生成 DataAdapter 的 Command 命令，否则，就要自己给 InsertCommand、UpdateCommand、DeleteCommand 属性定义 SQL 更新语句；使用 DataAdapter 对象的 Fill 方法把 Select 语句的查询结果放在 DataSet 对象的数据表中；接下来，将该数据表复制到 DataTable 对象中；最后，修改 DataTable 对象中的数据，并通过 DataAdpter 对象的 Update 方法向数据库提交修改数据。





图 6-10　DataAdapterInsertDemo.aspx 页面设计及运行效果

【示例 6-8】 演示使用 DataSet 对象和 DataAdapter 对象修改数据库记录。

（1）在网站项目 ch06 中添加一个名为 DataAdpterUpdateDemo.aspx 的页面。

（2）编写 DataAdpterUpdateDemo.aspx 页面中“修改”按钮的 Click 事件方法的程序代码，如表 6-23 所示。

表 6-23　“修改”按钮的 Click 事件方法的程序代码

```
protected void btnUpdate_Click(object sender, EventArgs e)
{
    //从 web.config 中读取连接字符串
    string strCnn = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    DataSet ds = new DataSet();
    if (FileUpload1.HasFile == true) //将照片上传到网站的"images"文件夹中,以学号为名
                                     //字进行保存
    {
        string fileName = this.txtStuNo.Text + ".jpg";
        FileUpload1.SaveAs(Server.MapPath(("~/images/") + fileName));
    }
    using (SqlConnection cnn = new SqlConnection(strCnn))  //创建连接对象
    {
        //创建 DataAdapter 对象,使用 select 语句和连接对象初始化
        SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", cnn);
        //建立 CommandBuilder 对象来自动生成 DataAdapter 对象的 Command 对象
        SqlCommandBuilder sb = new SqlCommandBuilder(sda);
        sda.Fill(ds, "StuTable");  //调用 Fill 方法,填充 DataSet 的数据表 StuTable
        DataTable dtable = ds.Tables["StuTable"];  //将数据表 StuTable 的数据复制到
                                                   //DataTable 对象
```

续表

代　码　页

```
            //设置 dtable 的主键,以便用后面调用 Find 方法查询记录
            dtable.PrimaryKey = new DataColumn[] { dtable.Columns["StuNo"] };
            //根据 txtStuNo 文本框的输入查询相应的记录,以便修改
            DataRow dr = dtable.Rows.Find(txtStuNo.Text.Trim());
            if (dr != null) //如果存在相应记录,则修改并更新到数据库
            {
                dr.BeginEdit(); //修改记录值开始
                dr["StuNo"] = txtStuNo.Text.Trim();
                dr["Name"] = txtName.Text.Trim();
                dr["Sex"] = DdSex.SelectedValue;
                dr["Birth"] = Convert.ToDateTime(txtBirth.Text.Trim());
                dr["Photo"] = txtStuNo.Text.Trim() + ".jpg";
                dr.EndEdit();
                sda.Update(dtable);     //提交更新
                Response.Write("修改成功!请打开数据库查看");
            }
            else
            {
                Response.Write("该学生不存在!");
            }
        }
    }
```

(3) 程序运行效果如图 6-11 所示。





图 6-11　DataAdpterUpdateDemo.aspx 页面设计及运行效果

4) 删除数据库的数据

使用 DataSet 对象和 DataAdapter 对象删除数据库数据的一般步骤如下:首先建立数

据库连接；然后利用数据连接和 Select 语句建立 DataAdapter 对象；定义 DeleteCommand 属性，自定义 Delete 命令；使用 DataAdapter 对象的 Fill 方法把 Select 语句的查询结果放在 DataSet 对象的数据表中；接下来，将该数据表复制到 DataTable 对象中；最后，删除 DataTable 对象中的数据，并通过 DataAdpter 对象的 Update 方法向数据库提交数据。

【示例 6-9】 演示使用 DataSet 对象和 DataAdapter 对象删除符合条件的记录。

（1）在网站项目 ch06 中添加一个名为 DataAdpterDeleteDemo.aspx 的页面。

（2）编写 DataAdpterDeleteDemo.aspx 页面中“删除”按钮的 Click 事件方法的程序代码，如表 6-24 所示。

表 6-24 “删除”按钮的 Click 事件方法的程序代码

行号　　代　码　页

```
protected void btnDel_Click(object sender, EventArgs e)
{
  string strCnn = ConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
  DataSet ds = new DataSet();
  using (SqlConnection cnn = new SqlConnection(strCnn))
  {
    SqlDataAdapter sda = new SqlDataAdapter("select * from StuInfo", cnn);
    //定义 DeleteCommand 属性,自定义 Delete 命令,其中@StuNo 是参数
    sda.DeleteCommand = new SqlCommand("delete from StuInfo where StuNo = @StuNo", cnn);
    //定义@StuNo 参数对应于 StuInfo 表的 StuNo 列
    sda.DeleteCommand.Parameters.Add("@StuNo", SqlDbType.VarChar, 8, "StuNo");
    //调用 Fill 方法,填充 DataSet 的数据表 StuTable
    sda.Fill(ds, "StuTable");
    //将数据表 StuTable 的数据复制到 DataTable 对象
    DataTable dtable = ds.Tables["StuTable"];
    //设置 dtStuInfo 的主键,便于后面调用 Find 方法查询记录
    dtable.PrimaryKey = new DataColumn[] { dtable.Columns["StuNo"] };
    //根据 txtStuNo 文本框的输入查询相应的记录,以便修改
    DataRow dr = dtable.Rows.Find(txtStuNo.Text.Trim());
    //如果存在相应记录,则删除并更新到数据库
    if (dr != null)
    {
        dr.Delete();              //删除行记录
        sda.Update(dtable);       //提交更新
        Response.Write("记录删除成功!" + "< hr >");
    }
    else
    {
        Response.Write("没有该记录!" + "< hr >");
    }
  }
}
```

（3）程序运行效果如图 6-12 所示。





图 6-12 DataAdpterDeleteDemo. aspx 页面设计及运行效果

6.1.6 系统架构和分层

1. 系统框架

到目前为止，我们已经完成了“新知书店”项目的页面框架的搭建，从现在开始将学习三层架构在 ASP.NET 中的应用，理解多层结构思想在项目开发中的突出优势。

搭建什么样的系统框架取决于项目的具体需求。有的企业站点只是公司的介绍，数据库的内容可能就几篇新闻，在这种情况下，基本不需要做太多的系统设计工作，有一个通用的数据库访问类就足够了。有的站点非常庞大，比如新浪、网易等站点，这类项目往往需要很多部门的协作才能完成。而且由于功能模块复杂，程序员也有各自的分工：有负责用户管理模块的、有负责权限管理模块的。如果模块功能上有交叉，就有必要制定一个统一的标准，大家都按照一致的标准操作，这就有必要设计一个完善的系统框架。

高内聚、低耦合是系统架构设计的原则。高内聚是指每一层都有统一的职能，对外不公开；低耦合是指层与层之间相互独立。比如根据客户需求变动，要求将基于三层架构的 WinForms 应用程序 MySchool 项目改成 Web 应用程序，那么只需要重新编写 MySchool 项目的表示层，继续使用原来的业务逻辑层和数据访问层即可，而不需要全部重新开发。使用多层结构时，还必须注意要遵循不能跨层访问的原则。

2. 三层架构概述

在软件体系设计中，分层式结构最常见，也是最重要的一种结构。微软公司推荐的分层式结构一般分为三层，从下至上分别为数据访问层（DAL）、业务逻辑层（BLL）和表现层（UI）。三层架构（3-Tier Architecture）是基于模块化程序设计的思想，为实现分解应用程序的需求而逐渐形成的一种标准模式的模块划分方法。

三层架构的软件系统不必为了业务逻辑上的微小变化而修改整个程序，只需要修改业务逻辑层中的方法（函数）即可，增强了代码的可重用性，便于不同层次的开发人员之间的合作。只要遵循一定的标准就可以进行并行开发，最终将各个部分拼接到一起即可构成最终的应用程序。

释疑：所谓“分层”，就是将应用程序按照不同的功能划分成不同的模块并加以实现，其

中每一层实现应用程序一个方面的逻辑功能。

1）三层架构的分层式结构

三层架构包含数据访问层(DAL)、业务逻辑层(BLL)和表现层(UI)，它们各自的功能如下：

- 数据访问层(DAL)——负责对数据库的访问，主要实现对数据表的增、删、改、查操作。
- 业务逻辑层(BLL)——负责业务处理和数据传递。它包含了与核心业务相关的逻辑，实现业务规则和业务逻辑。业务逻辑层处于数据访问层与表示层之间，还作为表示层和数据访问层的桥梁，实现数据的传递和处理，起到了数据交换中承上启下的作用。对于数据访问层而言，它是调用者；对于表示层而言，它却是被调用者。
- 表示层(UI)——负责内容的展示和与用户的交互。它位于最外层(最上层)，离用户最近，给予用户直接的体验。通俗地讲，就是展现给用户的界面。该层主要完成两个任务：从业务逻辑层获取数据并显示；与用户进行交互，将相关数据送回业务逻辑层进行处理。

分层的目的是为了体现"高内聚、低耦合"的思想。分层的时候如果没有一个适当的数据容器来贯穿各层，将导致耦合性过高，所以用模型层(Model)作为层与层之间数据传递的载体。模型层(Model)是标准和规范，它包含了与数据库表相对应的实体类。如图6-13所示为各层之间的关系。

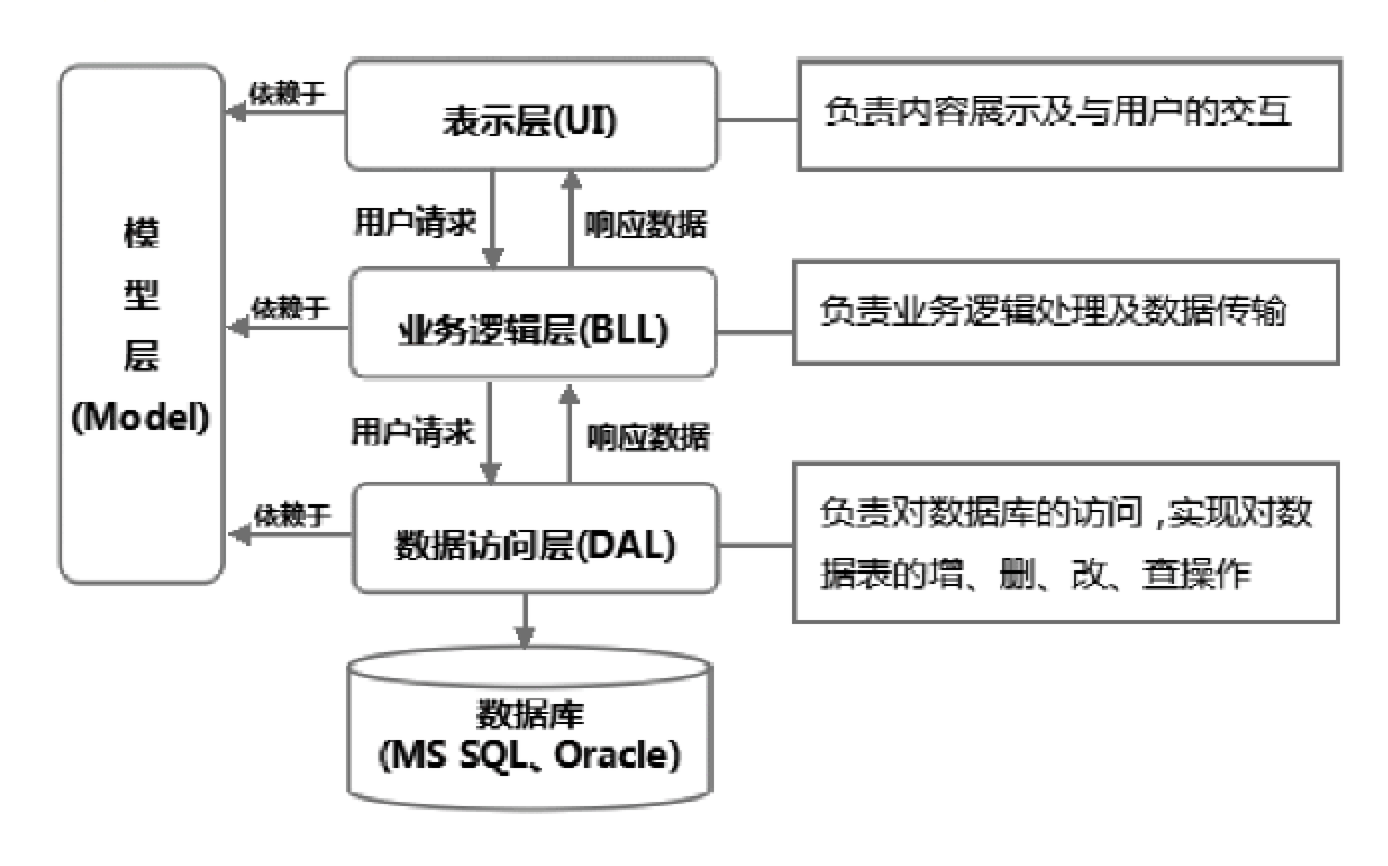


图6-13 三层的分层式结构

2）三层架构的优缺点

优点：

- 开发人员可以只关注整个结构中的某一层。
- 可以很容易用新的实现来替换原有层次的实现。
- 可以降低层与层之间的依赖。
- 有利于标准化。
- 利于各层逻辑的复用。
- 在后期维护的时候，极大地降低了维护成本和维护时间。

缺点：

- 降低了系统的性能。这是不言而喻的。如果不采用分层式结构，很多业务可以直接造访数据库，以此获取相应的数据，如今却必须通过中间层来完成。
- 有时会导致级联的修改。这种修改尤其体现在自上而下的方向。如果在表示层中需要增加一个功能，为保证其设计符合分层式结构，可能需要在相应的业务逻辑层和数据访问层中都增加相应的代码。
- 增加了开发成本。

6.2　单元任务

任务 6-2-1　实现“新知书店”管理员登录功能

【任务描述】

在任务 4-2-4 的基础上结合任务 5-2-6 基于母版页创建的后台首页。完成“新知书店”系统管理员登录功能，实现用户名和密码的非空验证，如果都不为空，则进行用户名和密码的数据验证（本任务的用户名和密码基于 BookShopPlus 的 Users 数据表），输入错误给出“用户名式密码错误！”的提示信息。

【任务实施】

本任务的实施较简单，前台登录页 AdminLogin. aspx 在任务 4-2-4 中已经全部设计完成，后台页面默认首页 Default. aspx 在第 5 单元的相关任务中设计完成。与任务 4-2-4 不同的是，本任务的用户名和密码不固定，而是要从数据库中进行检索。本任务的实施步骤如下：

（1）在任务 4-2-4 的基础上，创建解决方案下的网站项目 Web，结合任务 5-2-6 完成项目后台文件夹 Admin 下的文件的创建（其实这里可以把相应的页面文件及其对应的资源图片和样式表复制到本任务网站项目对应文件夹下）。

（2）右击网站项目 Web，选择“添加 ASP.NET 文件夹”—>App_Code 命令，在 App_Code 文件夹下创建数据表 Users 对应的实体类 Users. cs 和操作数据的类文件 DBOperator. cs，代码如下。

```
/// <summary>
///为简化操作,此实体类根据需要省略了表 Users 中除了用户名和密码外的其他字段
/// </summary>
public class Users
{
    public string LoginId { get; set; }                    //用户名
    public string LoginPwd { get; set; }                   //密码
}

  ⋮
/// <summary>
///DBOperator 类完成用户名和密码检查
```

```
/// </summary>
public class DBOperator
{
    //连接字符串
    private const string strConn = @"Data Source=.;Initial Catalog=BookShopPlus;
                                     Integrated Security=True";

    #region 检查用户和密码
    /// <summary>
    /// 检查用户和密码
    /// </summary>
    /// <param name="userName">用户名</param>
    /// <param name="userPwd">密码</param>
    /// <param name="strMsg">返回的登录信息</param>
    /// <returns>-2:数据库异常;-1:用户名或密码错误;0:用户名或密码为空;1:登录成
    /// 功</returns>
    public static int CheckUserInfo(string userName, string userPwd, ref string strMsg)
    {
        int Ret = -2;
        try
        {
            if (userName.Equals(string.Empty) || userPwd.Equals(string.Empty))
            {
                strMsg = "请输入用户名和密码";
                Ret = 0;
            }
            else
            {
                using (SqlConnection conn = new SqlConnection(strConn))
                {
                    conn.Open();
                    StringBuilder sb = new StringBuilder();
                    sb.AppendLine("SELECT");
                    sb.AppendLine("            *");
                    sb.AppendLine("FROM");
                    sb.AppendLine("          [Users]");
                    sb.AppendLine("WHERE");
                    sb.AppendLine("           LoginId='" + userName + "'");
                    sb.AppendLine("AND");
                    sb.AppendLine("           LoginPwd='" + userPwd + "'");
                    sb.AppendLine("AND UserRoleId=3");        //管理员的UserRoleId为"3"
                    SqlCommand comm = new SqlCommand(sb.ToString(), conn);
                    SqlDataReader reader = comm.ExecuteReader();
                    if (reader.Read())
                    {
                        Ret = 1;
                    }
                    else
                    {
                        Ret = -1;
```

```
                        strMsg = "用户名或密码错误!";
                    }
                }
            }
            return Ret;
        }
        catch (Exception ex)
        {
            strMsg = ex.Message;
            Ret = -2;
            return Ret;
        }
    }
}
```

(3) 将任务 4-2-4 中登录页后置代码文件 AdminLogin.aspx.cs 中固定用户名与密码的如下代码：

```
if (this.txtLoginId.Text.Trim() == "admin" && this.txtLoginPwd.Text.Trim() == "123456")
```

替换成如下两行即可：

```
int iRet = DBOperator.CheckUserInfo(this.txtLoginId.Text.Trim(), this.txtLoginPwd.Text.
Trim(),ref strMsg);
if (iRet>0)
```

任务 6-2-2　搭建“新知书店”系统三层架构

【任务描述】

在已完成的“新知书店”基础上，结合本单元所讲授的三层架构的概念，完成“新知书店”系统三层架构的搭建。

“新知书店”系统目录结构如图 6-14 所示。

【任务实施】

(1) 创建解决方案 BookShop，运行 Visual Studio 2010，在其菜单栏中选择“文件”→“新建项目”命令，打开“新建项目”对话框，在“新建项目”对话框中，展开已安装的模板中的“其他项目类型”节点，选择“Visual Studio 解决方案”选项，在中间窗口选择“空白解决方案”选项，新建一个名为 BookShop 的空白解决方案，如图 6-15 所示。

(2) 添加模型层。右击解决方案 BookShop，在弹出快捷菜单中选择“添加”→“新建项目”命令，在“添加新项目”对话框中，选择“类库”选项，如图 6-16 所示，新建一个名为 BookShopModels 的类库。

(3) 按照步骤(2)的方法，依次新建名为 BookShopDAL 的数据访问层、BookShopBLL 业务逻辑层。

(4) 添加表示层。右击解决方案 BookShop，在弹出的快捷菜单中选择“添加”→“新建

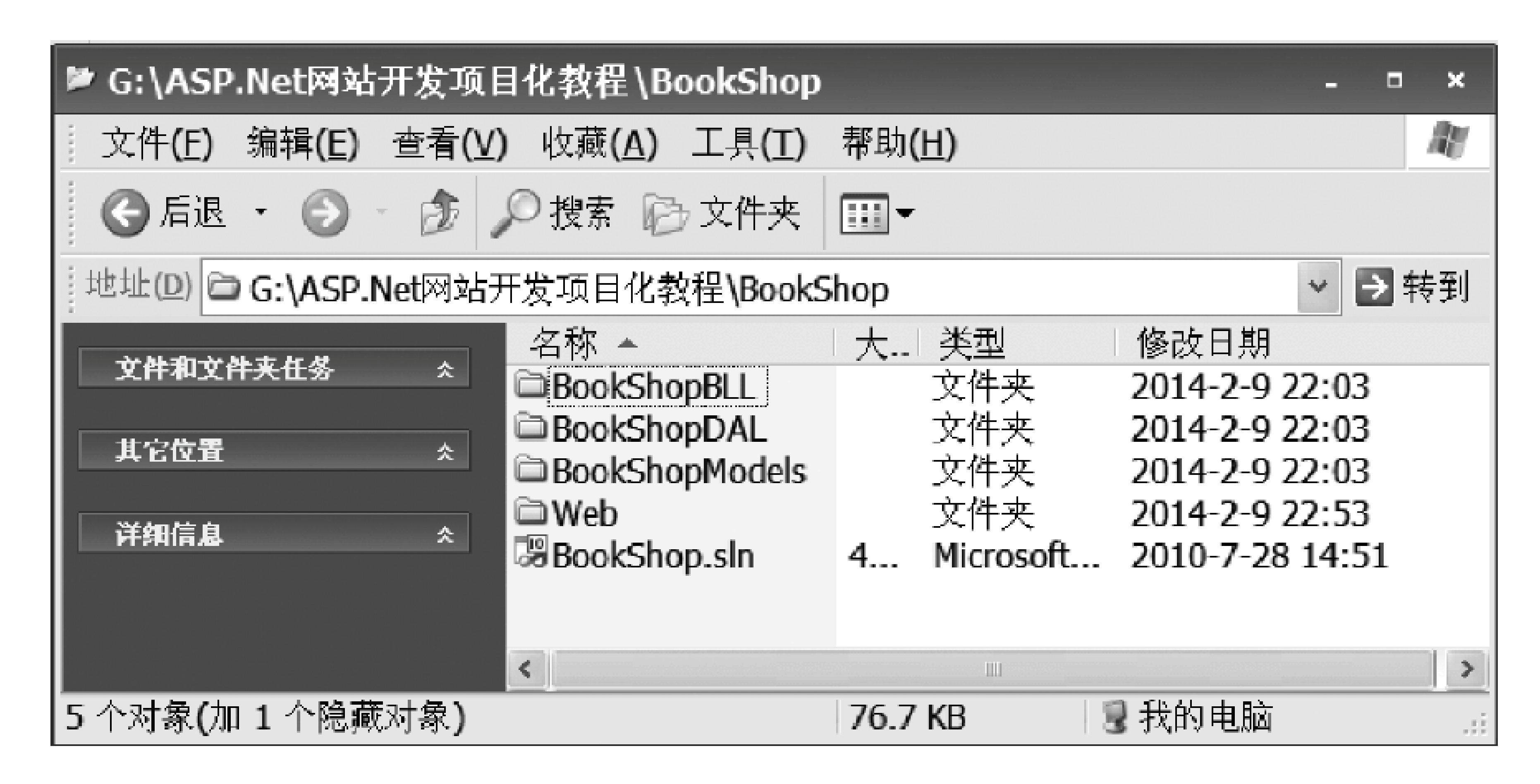


图 6-14 “新知书店”系统三层架构的文件结构



图 6-15 新建空白解决方案 BookShop

网站”命令。表示层的网站项目存放位置要选择解决方案所在文件夹，这里将表示层网站项目命名为 Web，如图 6-17 所示。

数据访问层一般命名为 DAL，或解决方案名+DAL，这里取名为 BookShopDAL；业务逻辑层一般取名为 BLL，这里取名为 BookShopBLL。

提示：*每一层的命名不是一成不变的，要根据实际项目需要综合考虑。*

经过刚才的 4 个步骤，基于三层结构的系统架构已经初步搭建成功。但每一层都是各自独立的，它们之间没有任何联系，因此需要给各层之间建立依赖关系。各层之间相互依赖

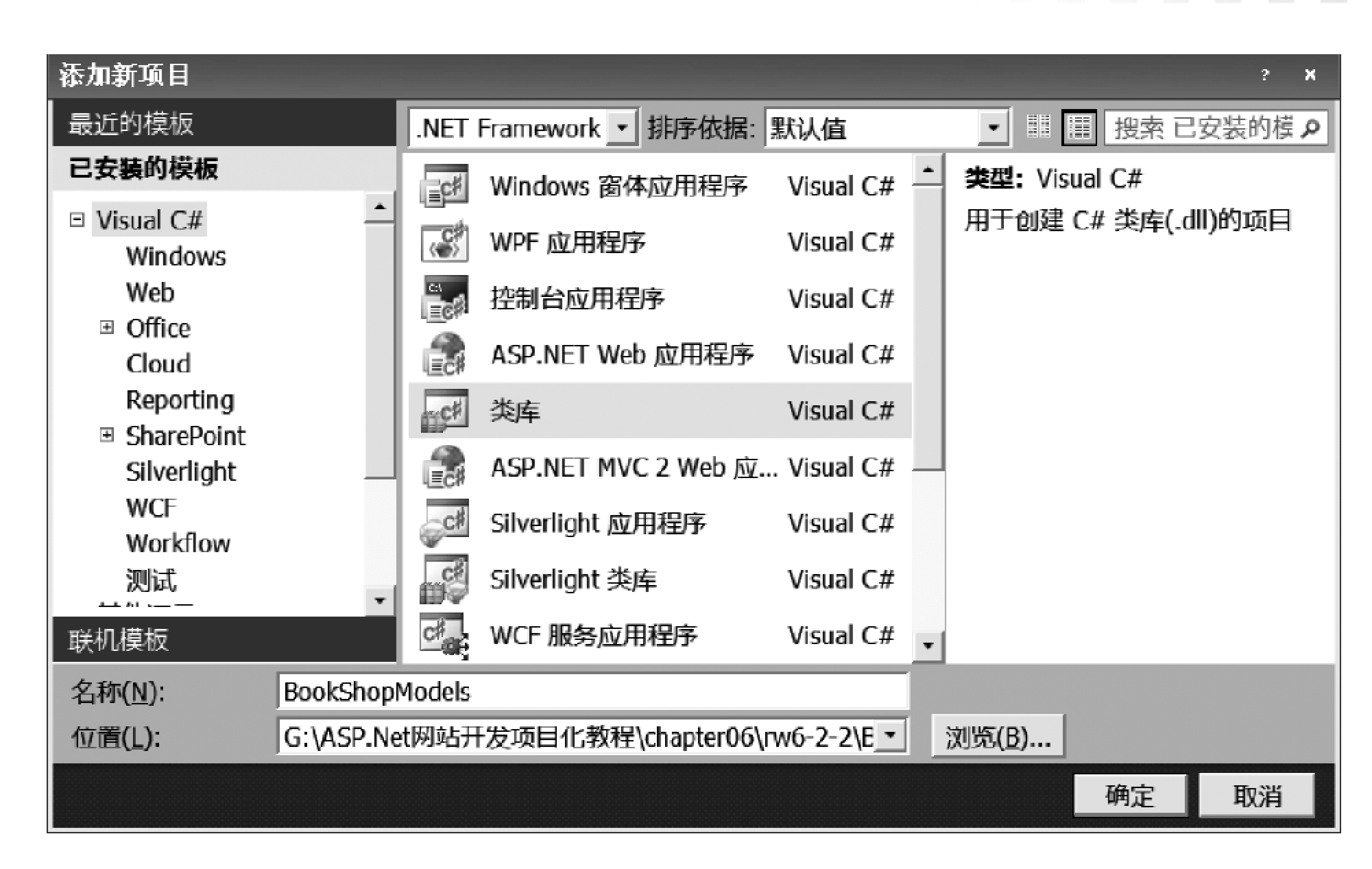


图 6-16　选择“类库”模板

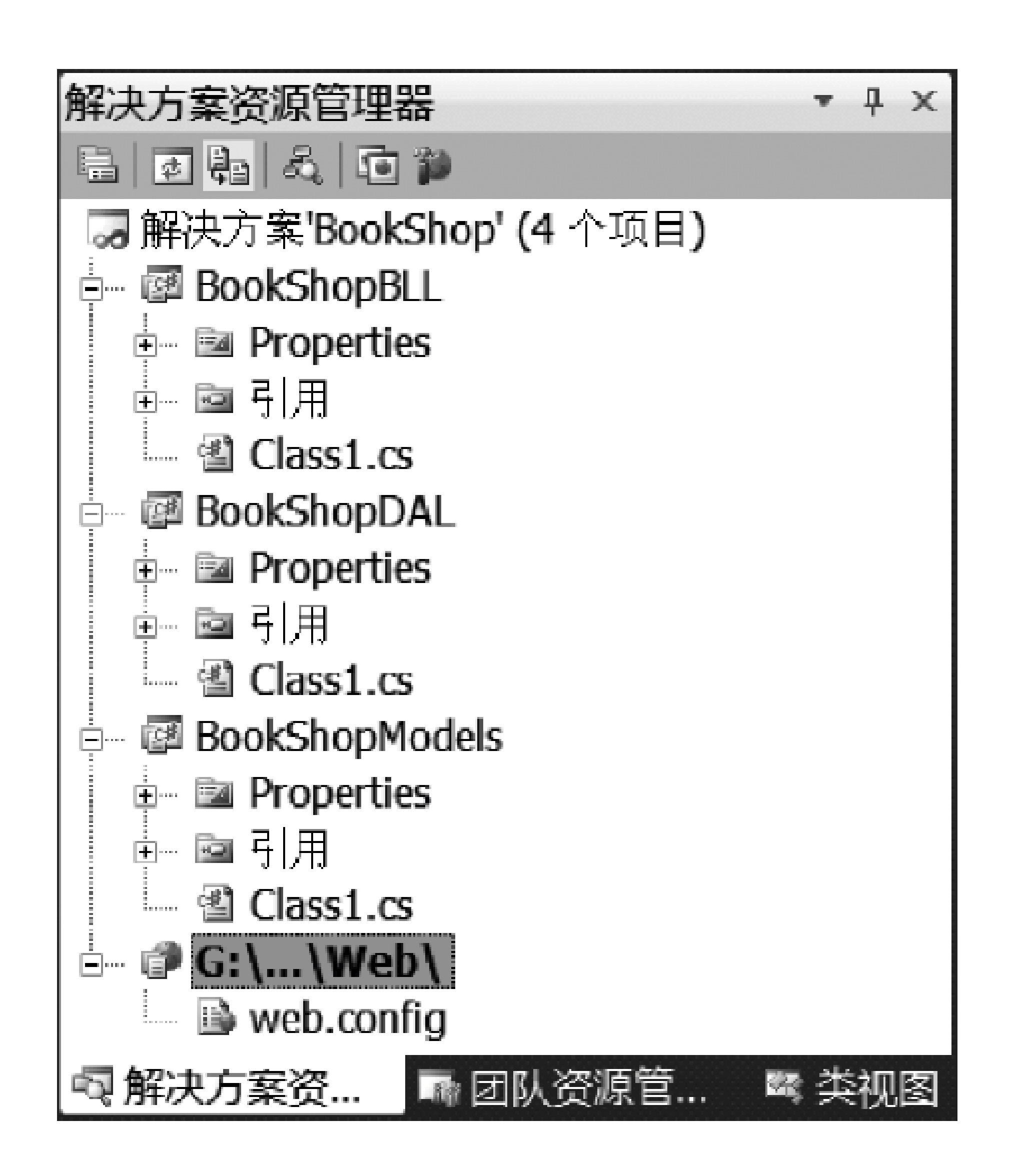


图 6-17　解决方案资源管理器效果

是它们良好协作的关键。

(5) 实现表示层对业务逻辑层的依赖。在“解决方案资源管理器”面板中，右击表示层(Web)，在弹出的快捷菜单中选择“添加引用”命令。在弹出的“添加引用”对话框中选择“项目”选项卡，选中项目名称 BookShopBLL，单击“确定”按钮，如图 6-18 所示。此时，在表示层的引用目录下就会出现业务逻辑层的项目名称，如图 6-19 所示。

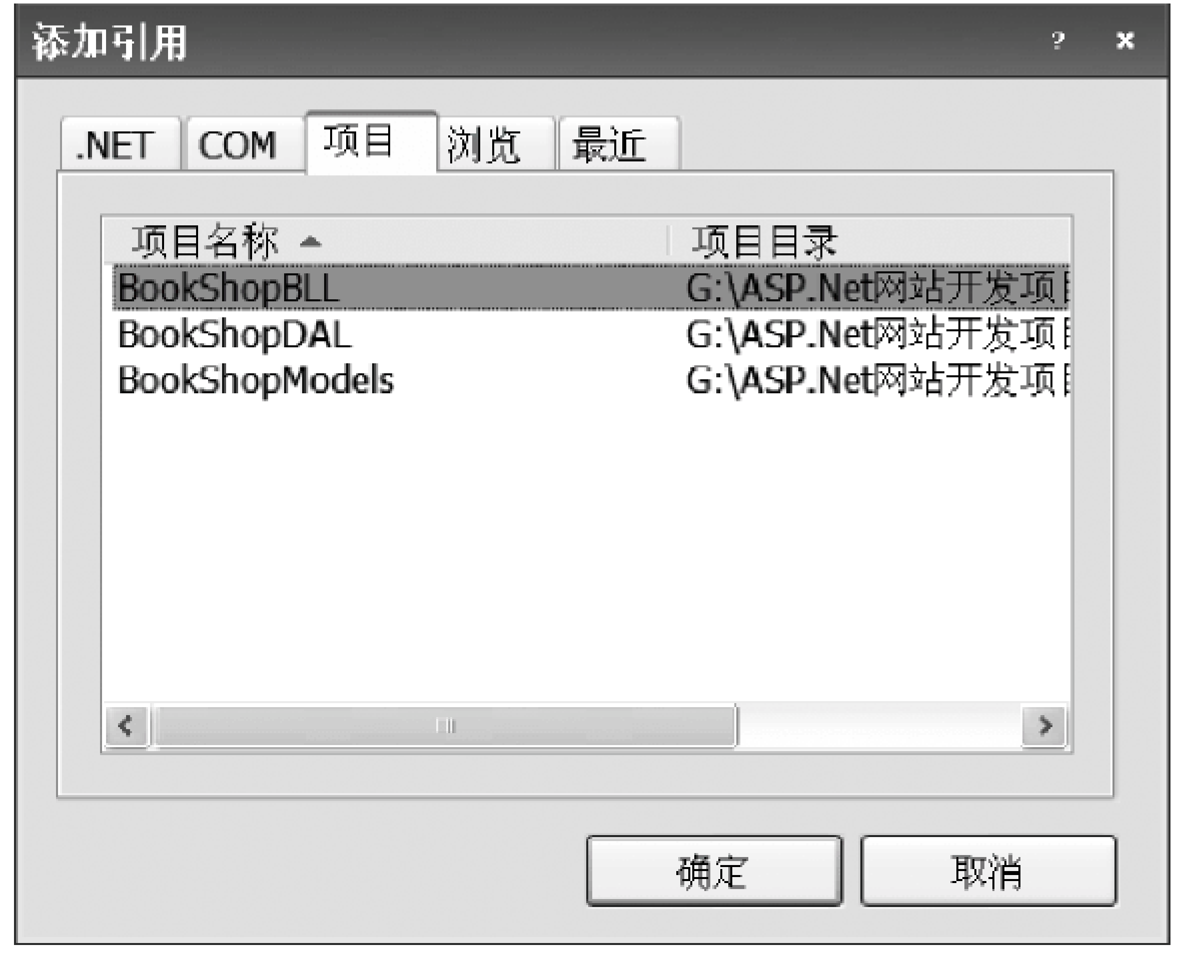



图 6-18 “添加引用”对话框

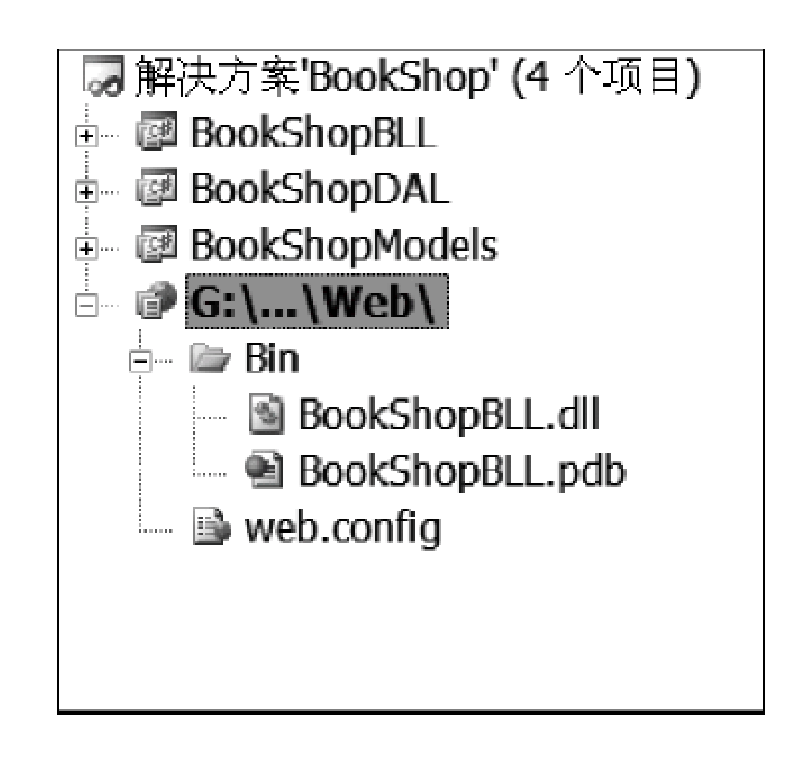


图 6-19 实现 Web 表示层的引用

(6) 实现业务逻辑层对数据访问层的依赖。即在业务逻辑层(BookShopBLL)实现对数据访问层(BookShopDAL)的引用,操作方法同步骤(5)。

(7) 实现数据访问层、业务逻辑层和表示层对数据模型层的依赖。由于数据访问层(BookShopDAL)、业务逻辑层(BookShopBLL)、表示层(Web)三层之间的数据传递的类型是基于模型层的实体类,所以这三层均需要添加对模型层(BookShopModels)的引用,操作步骤类似,在此不做详细讲解。至此,基于三层结构的“新知书店”系统架构才算真正搭建完成。

任务 6-2-3 实现三层架构下的“新知书店”用户注册功能

【任务描述】

- 在“新知书店”的三层架构下实现用户注册功能。
- 先判断用户是否存在。如果存在,就显示“用户名已存在,请重新输入!”的对话框;否则,将用户信息添加到 User 表中。

【任务实施】

1. 用户注册相关实体类的实现

实体类统一放在模型层,与数据库中的表相对应,是描述一个业务实体的“类”。整个应用系统业务所涉及的对象(例如“新知书店”中的图书、用户等)都可以看成是业务实体类。从数据存储的角度来看,业务实体类就是存储应用系统信息的数据表。将每一个数据表中的字段定义成属性,并将这些属性用一个类封装——这个类就称为“实体类”,三层之间的数据传递就是通过传输实体对象来达到目的的。

(1) 类的命名。模型层中实体类的命名一般和所对应的表名一致。数据表中有些表名以“s”结尾,比如“新知书店”中的用户表 Users 是复数形式,但实体类一般以单数形式 User 表示。

（2）实体类比较简单，根据数据库中的字段编写对应的变量和属性即可。除了构造方法以外，实体类一般没有其他方法(很多情况下把构造方法也省略，但不建议这么做）。

（3）外键的处理。根据第一单元的需求分析可知，用户有状态之分（无效、有效），也有角色之分（普通用户、VIP 或者管理员），这些内容在数据库中表现为外键关系。如何处理这些关系呢？一般有两种方式：使用外键表 ID 或者使用外键对象。使用外键表 ID 的方式比较简单，但采用外键对象的方式是当前非常流行的方式，许多开源的代码体系中都采用外键对象的方式处理。这种方式的好处是，它可以依据外键类直接访问外键的其他属性。比如“新知书店”的用户有角色之分（UserRole），使用外键类就可以用以下方式访问角色名称。

```
user.UserRole.Name
```

其中，user 表示用户表对应的实体类对象。

（4）用户注册相关实体类的编写。“新知书店”用户表有两个外键表（UserStates 和 UserRoles），如图 6-20 所示。

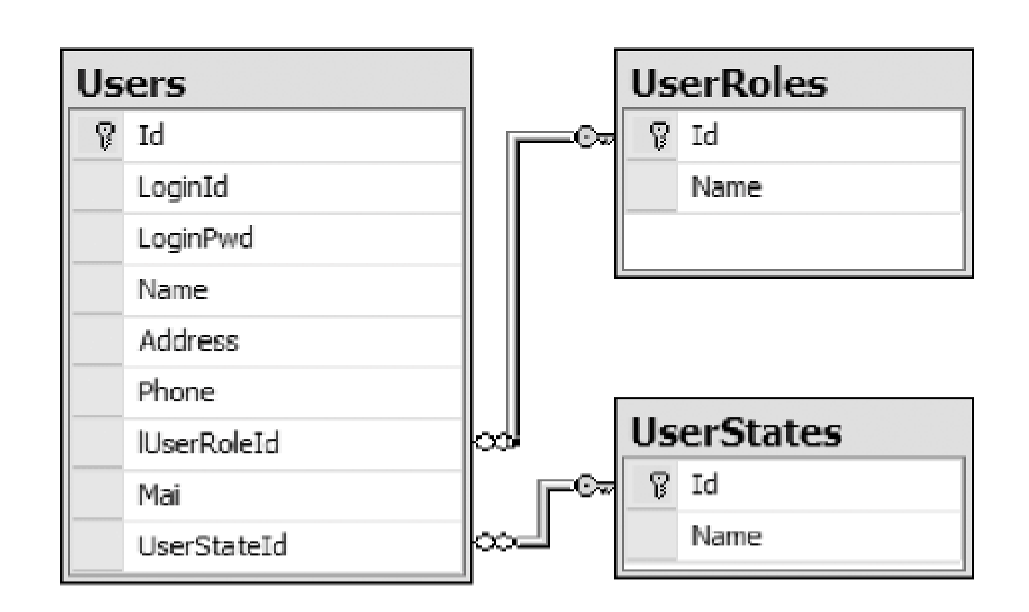


图 6-20　用户表及其外键表

根据如图 6-20 所示的用户表及其外键表，需要创建三个对应的实体类，如图 6-21 所示。







图 6-21　用户类、用户角色类及用户状态类

在项目 BookShopModels 中分别新建类文件 User.cs、UserRole.cs 和 UserState.cs。UserRole.cs 的实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BookShop.Models
{
    [Serializable()]        //将实体类标记为可序列化,保证数据在不同途径中传递的正确性
    public class UserRole
    {
        private int id;
        private string name = String.Empty;
        public UserRole() { }
        public int Id        //角色编号
        {
            get { return this.id; }
            set { this.id = value; }
        }
        public string Name   //角色名称
        {
            get { return this.name; }
            set { this.name = value; }
        }
    }
}
```

UserState.cs 的实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BookShop.Models
{
    [Serializable()]
    public class UserState
    {
        private int id;
        private string name = String.Empty;
        public UserState() { }
        public int Id                    //用户状态编号
        {
            get { return this.id; }
            set { this.id = value; }
        }
        public string Name               //用户状态名称
        {
            get { return this.name; }
            set { this.name = value; }
        }
    }
}
```

User.cs 的实现代码如下：

```
using System;
using System.Collections.Generic;
using System.Text;
namespace BookShop.Models
{
    [Serializable()]
    public class User
    {
        private int id;
        private UserState userState;
        private UserRole userRole;
        private string loginId = String.Empty;
        private string loginPwd = String.Empty;
        private string name = String.Empty;
        private string address = String.Empty;
        private string phone = String.Empty;
        private string mail = String.Empty;
        public User() { }
        public int Id
        {
            get { return this.id; }
            set { this.id = value; }
        }
        public UserState UserState     //用户状态,采用用户状态类作为 UserState 属性的类型
        {
            get { return this.userState; }
            set { this.userState = value; }
        }
        public UserRole UserRole       //用户角色
        {
            get { return this.userRole; }
            set { this.userRole = value; }
        }
        public string LoginId          //登录名
        {
            get { return this.loginId; }
            set { this.loginId = value; }
        }

        public string LoginPwd         //登录密码
        {
            get { return this.loginPwd; }
            set { this.loginPwd = value; }
        }

        public string Name             //真实姓名
        {
            get { return this.name; }
```

```
                set { this.name = value; }
            }

            public string Address       //地址
            {
                get { return this.address; }
                set { this.address = value; }
            }

            public string Phone         //电话号码
            {
                get { return this.phone; }
                set { this.phone = value; }
            }

            public string Mail          //邮件地址
            {
                get { return this.mail; }
                set { this.mail = value; }
            }
        }
    }
```

2. 用户注册数据访问层的实现

数据访问层封装了与数据库交互的操作,包括对数据表的增、删、改、查操作(C:Create、D:Delete、U:Update、R:Retrieve),数据访问层就针对每个数据表提供增、删、改、查操作,不做业务逻辑的判断。

使用ADO.NET连接数据库需要编写固定格式的代码,比如使用SqlConnection对象实现数据连接,使用SqlCommand对象执行SQL命令,使用SqlDataReader读取数据,使用DataSet结合SqlDataAdapter实现断开式数据库操作等。在前面的内容中,我们要在每一个数据访问层的方法中编写重复的ADO.NET代码,本节使用一个封装了ADO.NET方法的类——SqlHelper类,来提高数据访问代码的可重用性。

(1) 针对模型层中的每一个类,数据访问层有一个对应的数据访问类。比如针对User实体类,有一个对应的UserService类,负责有关Users表的数据处理。

(2) 数据库操作方法分析。刚才提到,数据访问层就是处理数据的增、删、改、查,所以编写的UserService类中的处理方法也围绕在四个方面。

增(Create)——一般增加的方法很简单,针对用户类的增加方法如下:

```
public void AddUser(User user)
```

删(Delete)——删除方法如下,一般根据Id执行删除操作:

```
public  bool DeleteUserById(int id)
```

改(Update)——修改的方法也很单一,典型的修改方法如下:

```
public  bool ModifyUser(User user)
```

查(Retrieve)——典型的查询方法如下:

```
public  List<User> GetUsers()
public  User GetUserById(int id)
public  User GetAdminUserByLoginId(string loginId)
private  List<User> GetUsersBySql(string safeSql)
```

在返回多个结果(结果集合)时,可以使用泛型集合 List<User>的形式,它是 User 对象列表,当访问用户姓名这个字段时,可以写为:

```
string name = user[0].Name;        //假设返回的对象集合为 user
```

针对 Id 进行的查询是非常常见的查询方法,而针对登录名进行的查询是用户类特有的(在其他类中,如果有 Name 属性,可能会有针对 Name 的查询)。需要注意的是,通过 SQL 语句查询的方法 GetUsersBySql,由于需要 SQL 语句作为参数,不能在业务逻辑层调用此方法,所以设置为 private 访问权限。数据访问层就是增、删、改、查,它不需要进行业务逻辑的判断。

(3) 过去进行数据库的访问操作都是非常程式化的步骤:创建连接对象→打开链接→执行 SQL 语句或存储过程→返回结果→关闭连接。每次都编写重复的代码,不同的部分只有 SQL 语句。为避免出现这种情况,将比较程式化的内容提取出来编写成 SqlHelper 类。

SqlHelper 类中应该包含常用的对数据库操作的方法,如图 6-22 所示是 SqlHelper 的类图。

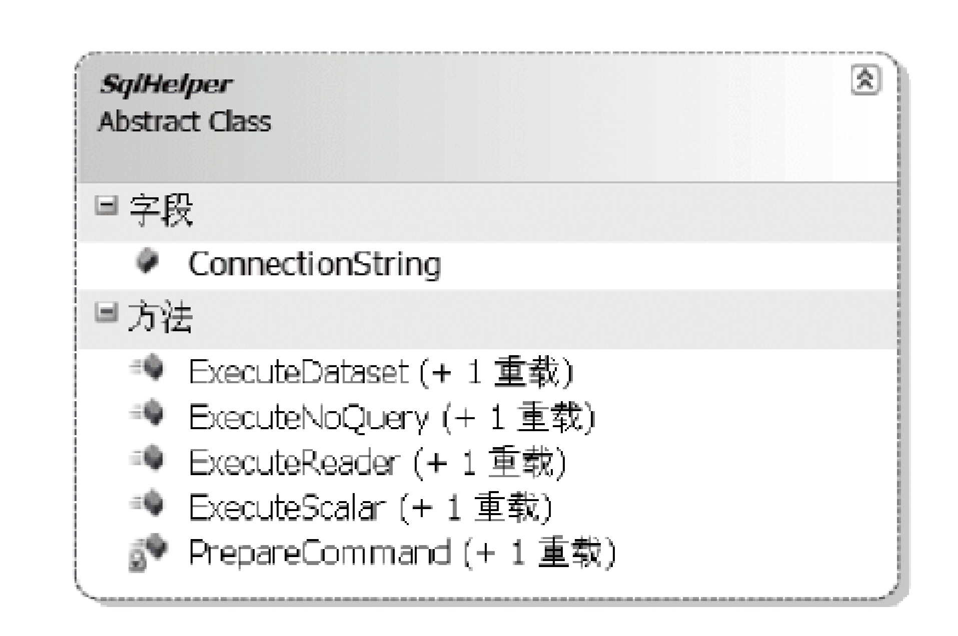


图 6-22　SqlHelper 类的类图

这里 SqlHelper 类中只用到了图 6-22 中的几个方法。为了今后便于扩展以及提高代码的重用率,将 SqlHelper 类定义成抽象的,该类中有连接属性,另外,还有一些数据库的操作方法。

- ExecuteNoQuery()方法是执行 SQL 语句或者存储过程后,返回受影响的行数。
- ExecuteScalar()方法也是执行 SQL 语句或者存储过程后,返回第一行的第一列,比如插入新记录,需要返回自增的 ID。
- ExecuteReader()方法也是执行 SQL 语句或者存储过程后,返回一个 DataReader。
- ExecuteDataSet()方法是执行 SQL 语句或者存储过程后,返回一个 DataSet。
- PrepareCommand()方法是构建一个 Command 对象供类的内部方法调用。它有两个重载方法,其中传 params object[] parameterValues 类型参数的方法会自动获取存储过程的参数名,只需要传参数值即可,使用起来非常方便。

(4) 用户注册数据访问层的编写,在 BookShopDAL 项目中新建类文件 UserService.cs、UserStateService.cs 和 UserRoleService.cs,限于篇幅,这里只列出 UserService.cs 的代码。

下面介绍 UserService.cs 类中重要方法的实现代码。

增——添加新用户(AddUser),返回添加后的用户对象,代码如下:

```
/// <summary>
/// 添加新用户,返回添加后的用户对象
/// </summary>
/// <param name="user"></param>
/// <returns></returns>
public void AddUser(User user)
{
    string sql  = "INSERT Users (LoginId, LoginPwd, Name, Address, Phone, Mail, UserRoleId,
                  UserStateId)" + "VALUES (@LoginId, @LoginPwd, @Name, @Address, @Phone,
                  @Mail, @UserRoleId, @UserStateId)";        //拼接 SQL 语句
    sql += " ; SELECT @@IDENTITY";                            //获取最新添加的用户 ID
    SqlParameter[] para = new SqlParameter[]
    {
        new SqlParameter("@UserStateId", user.UserState.Id), //FK(外键)
        new SqlParameter("@UserRoleId", user.UserRole.Id),   //FK
        new SqlParameter("@LoginId", user.LoginId),          //将 user 对象的属性作为参数
        new SqlParameter("@LoginPwd", user.LoginPwd),
        new SqlParameter("@Name", user.Name),
        new SqlParameter("@Address", user.Address),
        new SqlParameter("@Phone", user.Phone),
        new SqlParameter("@Mail", user.Mail)
    };
    user.Id = Convert.ToInt32(SqlHelper.ExecuteScalar(this.connection, CommandType.Text,
sql, para));
}
```

删——一般情况下,删除也是针对单一用户的操作,根据 Id 删除(DeleteUserById)的代码如下:

```
/// <summary>
/// 根据 id 删除用户
/// </summary>
/// <param name="id"></param>
public bool DeleteUserById(int id)
{
    string sql = @"DELETE OrderBook WHERE OrderID IN(SELECT Orders.Id FROM Orders
                   INNER JOIN Users ON Orders.UserId = Users.Id WHERE UserId = @Id)
                   DELETE Orders where UserId = @Id
                   DELETE Users WHERE Id = @Id";
    SqlParameter[] para = new SqlParameter[]
    {
      new SqlParameter("@Id", id)
    };
    return SqlHelper.ExecuteNoQuery(this.connection, CommandType.Text, sql, para) > 0;
}
```

改——修改用户状态(ModifyUserState)的代码如下：

```
/// <summary>
/// 更改会员状态
/// </summary>
/// <param name="id"></param>
/// <param name="status"></param>
public bool ModifyUserState(int id, UserStates state)   //以用户状态为参数
{
    //执行拼接的 SQL 语句,更新数据库对应列
    string sql = "Update users SET userstateid = " + Convert.ToByte(state) + " WHERE Id =
@UserId";
    return SqlHelper.ExecuteNoQuery(this.connection, CommandType.Text, sql,
                                                            new SqlParameter("@UserId",
id)) > 0;
}
```

查——以执行 SQL 语句查询(GetUsersBySql)和查询所有用户(GetUsers)的方法代码如下：

```
/// <summary>
/// 依据 sql 语句查询用户
///private 方法,提供用于执行的 SQL 语句并返回 User 对象集合的方法,该方法不对业务逻辑层
///公开,仅供数据访问层内部调用
/// </summary>
/// <param name="safeSql"></param>
/// <returns></returns>
private List<User> GetUsersBySql(string safeSql)
{
    List<User> list = new List<User>();
    DataSet ds = SqlHelper.ExecuteDataset(this.connection, CommandType.Text, safeSql);
    if (ds.Tables.Count > 0)
    {
        DataTable dt = ds.Tables[0];
        foreach (DataRow row in dt.Rows)
        {
            User user = new User();
            user.Id = (int)row["Id"];
            user.LoginId = (string)row["LoginId"];
            user.LoginPwd = (string)row["LoginPwd"];
            user.Name = (string)row["Name"];
            user.Address = (string)row["Address"];
            user.Phone = (string)row["Phone"];
            user.Mail = (string)row["Mail"];
               user.UserState = new UserStateService().GetUserStateById((int)row["
UserStateId"]);                             //FK
                user.UserRole = new UserRoleService().GetUserRoleById((int)row["
UserRoleId"]);                              //FK
            list.Add(user);
```

```
            }
        }
        return list;
    }

    /// <summary>
    /// 查询所有用户
    /// </summary>
    /// <returns></returns>
    public List<User> GetUsers()
    {
        string sqlAll = "SELECT * FROM Users";          //返回所有用户的 SQL 语句
        return GetUsersBySql(sqlAll);
    }
```

上面的增、删、改、查在本任务中只会用到 AddUser()，其他方法列出是为了示范说明，当然，在以后的任务中会陆续用到。

3. 用户注册业务逻辑层的实现

业务逻辑层是表示层与数据访问层的桥梁，负责业务处理和数据传递。该部分的方法一般与实际需求相关，如“新知书店”的用户注册，实际上不仅仅是添加一条记录那么简单的事情，它还包含了验证登录名 LoginId 是否已存在等业务逻辑。

（1）类的命名。业务逻辑层里面的类也是与模型层中的类相对应的一系列类，一般命名为实体类＋Manager，如用户处理的类就命名为 UserManager。

（2）业务逻辑层里面类中方法的编写。业务逻辑层应该提供哪些方法一般根据实际需求来确定，比如页面上有用户登录的功能，就可以考虑在业务逻辑层创建一个对应的用户登录的方法（像 Login()这样的方法不建议出现在数据访问层，从理论上说，数据访问层应该只看到基本的 CRUD 操作）。

```
    /// <summary>
    /// 登录验证
    /// </summary>
    /// <param name = "loginId">登录名</param>
    /// <param name = "loginPwd">登录密码</param>
    /// <param name = "validUser">输出用户</param>
    /// <returns>返回 true 表示成功</returns>
    public bool LogIn(string loginId, string loginPwd, out User validUser)
    {
        User user = new UserService().GetUserByLoginId(loginId);//判断用户是否存在
        if (user == null)
        {
            validUser = null;                           //用户名不存在
            return false;
        }
        if (user.LoginPwd == loginPwd)                  //判断用户的密码是否输入正确
        {
```

```
            validUser = user;
            return true;
        }
        else
        {
            validUser = null;                   //密码错误
            return false;
        }
    }
```

这个方法根据表示层(Web)提交过来的用户名和密码,判断是否为合法用户,如果是合法用户,则返回 true。这个方法的形参 validUser 前使用了 out 关键字。当某个方法需要有多个返回值时,可以用它来传递返回值,out 在这里的作用是返回一个用户对象,即当用户合法时,将该用户对象返回,以备在表示层中调用,比如将该用户对象的相关信息存入 Session 或者 Cookie。

如果页面上还需要有用户注册功能,就在业务逻辑层再创建一个用户注册的方法,代码如下。

```
    /// <summary>
    /// 注册新用户
    /// </summary>
    /// <param name="user"></param>
    /// <returns></returns>
    public bool Register(User user)
    {
        if (LoginIdExists(user.LoginId))
        {
            return false;
        }
        else
        {
            AddUser(user);  //添加用户的方法
            return true;
        }
    }
```

注册方法返回一个布尔值,这是因为注册时需要对用户名进行重名判断,如果用户名重复,则返回注册失败提示。

上面的两个方法都是对业务逻辑处理,业务逻辑层还常常在表示层和数据访问层之间的传递数据,如要返回所有用户的列表,可以编写如下方法:

```
    /// <summary>
    /// 获得所有用户
    /// </summary>
    /// <returns></returns>
    public List<User> GetUsers()
    {
```

```
    return new UserService().GetUsers();
}
```

该方法没有业务逻辑上的处理，但是由于表示层不可以直接访问数据访问层的代码，所以该方法仅仅是调用了一下数据访问层的相关方法，为表示层提供所需要的数据。

鉴于业务逻辑层起着表示层和数据访问层之间的桥梁的作用，一般数据访问层公开的方法会在业务逻辑层有个相对应的方法。对于 GetUsers()方法，数据层是读出 Users 表中的所有记录，表示层是要展示一个所有用户的列表，而业务逻辑层只是它们中间的桥梁。

4. 用户注册表示层的实现

表示层用于显示数据和接收用户输入的数据，为用户提供一种交互式操作的界面，带给用户直接的体验。可以说前面的几层是基础，表示层是最终呈现。在 ASP.NET 中，表示层就是整个 Web 站点，具体的内容要根据需求的内容而来。"新知书店"有用户系统模块，自然就需要有相关的用户登录、注册和管理等页面；图书系统也就要有图书管理、图书列表和图书详细信息展示等页面；在线销售还需要购物车、订单管理等页面。

在表示层，如果仅仅供用户展现内容，可能只需要将控件绑定数据即可，不需要编写任何代码；如果需要和用户交互，就要编写相关的事件代码。比如在管理员登录页，管理员单击"登录"按钮的事件，就需要进行用户输入内容的非空验证，然后通过调用业务逻辑层的相关方法判断用户名和密码是否匹配，编写代码验证用户的身份是否为管理员等。

用户注册表示层除了将接收的用户注册信息传递给业务逻辑层处理外，还需要做一系列的注册信息验证，比如各种输入内容的非空验证、两次输入密码的比较验证、电子邮箱格式的合法性验证，以及为防止系统被恶意注册要求用户输入的验证码等。下面完成用户注册表示层的实现。

先在表示层 Web 项目中按任务 5-2-6 的步骤完成母版页 common.master 的设计，然后基于母版页创建用户注册内容页 Register.aspx，任务 3-2-3 已经编写过代码，在此不予赘述。

编写用户注册页面后置代码文件 Register.aspx.cs 的代码如下：

```
⋮
using BookShop.BLL;
using BookShop.Models;
public partial class Register : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!IsPostBack)
    {
        snCode.Create();
    }
  }
  protected void btnSubmit_Click(object sender, EventArgs e)
  {
```

```
        if (!CheckCode())
        {
            Page.RegisterClientScriptBlock("alert", "<script>alert('验证码错误!')</script>");
            return;
        }
        User user = new User();
        user.LoginId = this.txtLoginId.Text;
        user.LoginPwd = this.txtLoginPwd.Text;
        user.Name = this.txtName.Text;
        user.Address = this.txtAddress.Text;
        user.Phone = this.txtTele.Text;
        user.Mail = this.txtEmail.Text;
        UserManager manager = new UserManager();
        if (!manager.Register(user))
        {
            Page.RegisterClientScriptBlock("alert",
                                       "<script>alert('用户名已使用,请重新输入!')</script>");
        }
        else
        {
            Page.RegisterClientScriptBlock("alert",
                "<script>alert('注册成功,请登录!');window.location='../default.aspx'</script>");
        }
    }
    protected bool CheckCode()
    {
        if (snCode.CheckSN(txtCode.Text.Trim()))  //判断验证码输入是否正确
        {
            return true;
        }
        else
        {
            snCode.Create();                      //如果验证码输入不正确,则生成新验证码
            return false;
        }
    }
}
```

用户注册页运行效果如图6-23所示,输入注册信息,单击"确定了,马上提交"按钮后将提示注册成功提示信息。

任务6-2-4　实现三层架构下的"新知书店"用户登录功能

【任务描述】

在三层架构下,基于母版页创建如图6-24所示的用户登录页并实现登录功能。具体要求如下:

- 用户名和密码要有非空验证和数据验证,输入错误要给出对话框提示;输入正确将用户跳转到首页。



图 6-23 用户注册页面效果图

- 用户选中"在此计算机上保留我的密码"时，客户端保存用户信息；页面加载时，判断客户端保存的信息。如果已保存，则给出提示。
- 用户单击"还没有注册？"链接时，将跳转到注册页面。

【任务实施】

1. 用户登录数据访问层的实现

当用户登录系统时，将用户输入的登录名提交数据库进行查询，如果查询的登录名存在，返回一个用户对象，不存在则返回 null。

在 BookShopDAL 项目中的 UserService.cs 类中，定义一个根据登录名查询用户的方法，具体代码如下。

```
⋮
using BookShop.Models;
using System.Configuration;
namespace BookShop.DAL
{
public class UserService
{
```

```
        string connection = ConfigurationManager.ConnectionStrings["BookShop"].ConnectionString;
        /// <summary>
        /// 根据登录名查询用户
        /// </summary>
        /// <param name="loginId"></param>
        /// <returns></returns>
        public User GetUserByLoginId(string loginId)
        {
            string sql = "SELECT * FROM Users WHERE LoginId = @LoginId";
            int userStateId;
            int userRoleId;
            User user = null;
            using (SqlDataReader reader = SqlHelper.ExecuteReader(this.connection,
                                 CommandType.Text, sql, new SqlParameter("@LoginId", loginId)))
            {
                if (reader.Read())
                {
                    user = new User();

                    user.Id = (int)reader["Id"];
                    user.LoginId = (string)reader["LoginId"];
                    user.LoginPwd = (string)reader["LoginPwd"];
                    user.Name = (string)reader["Name"];
                    user.Address = (string)reader["Address"];
                    user.Phone = (string)reader["Phone"];
                    user.Mail = (string)reader["Mail"];
                    userStateId = (int)reader["UserStateId"]; //FK
                    userRoleId = (int)reader["UserRoleId"]; //FK
                    reader.Close();
                    user.UserState = new UserStateService().GetUserStateById(userStateId);
                    user.UserRole = new UserRoleService().GetUserRoleById(userRoleId);

                }
            }
            return user;
        }
    }
}
```

代码中的 if 语句先构造了一个 user 对象,其属性值依次被赋值为数据库中查找到的某条记录的各个字段值,然后将该 user 对象返回。

2. 用户登录业务逻辑层的实现

在业务逻辑层创建一个对应的用户登录方法,接收表示层(Web)提交过来的登录名和密码,判断是否为合法用户。如果是合法用户,返回 True,否则返回 false,具体实现代码在任务 6-2-3 讲解用户注册业务逻辑层的实现时已经编写过,方法为 public bool LogIn(string loginId, string loginPwd, out User validUser),此处不再列出。这个方法的形参 validUser

前使用了 out 关键字，要注意它的作用。

3. 用户登录表示层的实现

用户单击“登录”按钮的事件，就需要进行用户输入内容的非空验证，然后通过调用业务逻辑层的相关方法判断用户名和密码是否匹配，编写代码验证用户的身份。这里为了减轻服务器的负担，把前台的非空验证交由 JavaScript 编写的客户端代码来完成（如果读者没有系统学习过 JavaScript 相关知识，可以不予理会，只要采用前面介绍过的验证控件给予实现即可）。为了实现用户登录，首先在 Web 项目中基于母版页 common.master 创建用户登录内容页 UserLogin.aspx，代码如下。

```
<asp:Content ID = "Content1" ContentPlaceHolderID = "cphHeader" runat = "Server">
<link href = "Css/member.css" rel = "stylesheet" type = "text/css" />
</asp:Content>
<asp:Content ID = "Content2" ContentPlaceHolderID = "cphContent" runat = "Server">
  <script type = "text/javascript">//<script>代码块用于实现用户名、密码的客户端非空验证
    function ValidateForm() {
        var txtLoginId = document.getElementById('<% = txtLoginId.ClientID %>');
        var txtLoginPwd = document.getElementById('<% = txtLoginPwd.ClientID %>');
        if (txtLoginId.value == "") {
            alert('请输入用户名!');
            return false;
        }
        else if (txtLoginPwd.value == "") {
            alert("请输入密码!");
            return false;
        }
        return true;
    }
    document.forms[0].onsubmit = function () {
        if (ValidateForm() == false) {
            return false;
        }
        else {
            document.forms[0].submit();
        }
    }
  </script>
  <div id = "action_area" class = "member_form">
    <h2 class = "action_type"><img src = "Images/login_in.gif" alt = "会员登录" /></h2>
    <p class = "state">欢迎光临新知书店网站!<br />您可以使用新知书店的用户名,直接登录.</p>
    <p>
        <label> 用户名</label>
        <asp:TextBox ID = "txtLoginId" runat = "server" CssClass = "opt_input"></asp:TextBox>
    </p>
    <p>
```

```
            <label>密    码</label>
            <asp:TextBox ID="txtLoginPwd" runat="server"
                            TextMode="Password"CssClass="opt_input"></asp:TextBox>
        </p>
        <p class="form_sub"><input type="checkbox" name="" checked="checked" />
            在此计算机上保留我的密码</p>
        <p class="form_sub">
            <asp:Button runat="server" ID="btnLogin" CssClass="opt_sub" Text="登录"
TabIndex="1"
                OnClick="btnLogin_Click" />
        <a href="Register.aspx">还没有注册?</a></p>
    </div>
</asp:Content>
```

编写用户登录页面后置代码文件 UserLogin.aspx.cs 中单击“登录”按钮事件方法代码如下：

```
....
using BookShop.BLL;
using BookShop.Models;
public partial class UserLogin : System.Web.UI.Page
{
  const string strErrorUser = "用户名或密码不正确,请重新填写!";
  protected void btnLogin_Click(object sender, EventArgs e)
  {
    User user;
    UserManager manager = new UserManager();
    if (manager.LogIn(this.txtLoginId.Text, this.txtLoginPwd.Text, out user))
    {
        Session["CurrentUser"] = user;
        if (Request.QueryString["ReturnUrl"] != null)
        {
            Response.Redirect(Request.QueryString["ReturnUrl"].ToString());
        }
        Response.Redirect("~/default.aspx");
    }
    else
    {
        Page.ClientScript.RegisterClientScriptBlock(this.GetType (),"alert",
                                        "<script>alert('" + strErrorUser + "')
</script>");
    }
  }
}
```

用户登录页运行效果如图 6-24 所示，页面样式、外观控制代码参见配套的教学资源源代码。



图 6-24 “新知书店”用户登录页面

6.3 项目实训

1. 搭建“博客系统”三层架构

【需求说明】

- 参照任务 6-2-2，完成“博客系统”三层架构的搭建，模型层的实体类名与数据库中的表名相对应，效果如图 6-25 所示。

2. 实现三层架构下的“博客系统”登录功能

【需求说明】

- 在母版页中，用户输入用户名和密码，系统检测用户是否存在。如果存在，则转入首页；如果不存在，则提示错误信息。
- 当用户处于未登录状态，界面如图 6-26 所示，登录后如图 6-27 所示。



图 6-25 “博客系统”的三层架构



图 6-26 登录前

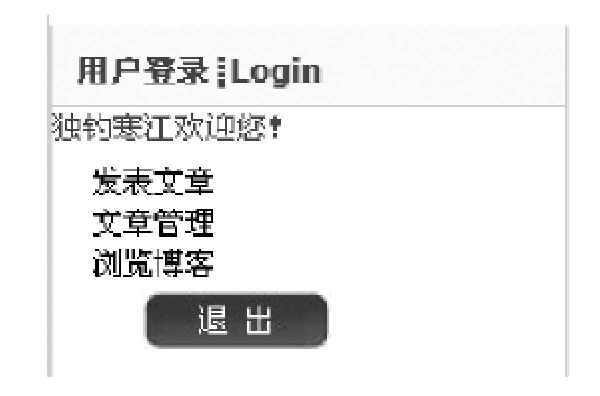


图 6-27 登录后

提示：编写各层中实现用户登录且验证的代码，登录 SQL 语句可以使用存储过程或参数化 SQL，登录成功后，用 Session 存储用户名和密码。

3. 实现三层架构下的“博客系统”会员注册功能

【需求说明】

输入如下信息，实现“博客系统”会员注册功能。

- 用户名：要求输入 6～18 位的字符，用户名只能由英文、数字及下划线组成。
- 密码：要求输入 6～18 位的字符，密码只能由英文、数字组成。
- 确认密码：与上面的密码保持一致。
- E-mail：必须包含@字符。
- QQ 和昵称。

6.4　单元小结

本单元介绍了 ADO.NET 的两个组成部分——.NET 数据提供程序和 DataSet 数据集，并着重讲解了 SqlCommand 对象、DataSet 数据集、SqlDataAdapter 对象 DataTable 数据表、DataReader 的使用与技巧，为更好地理解数据的操作打下坚实的基础；对三层架构进行了详细介绍与设计。本单元知识体系如图 6-28 所示。

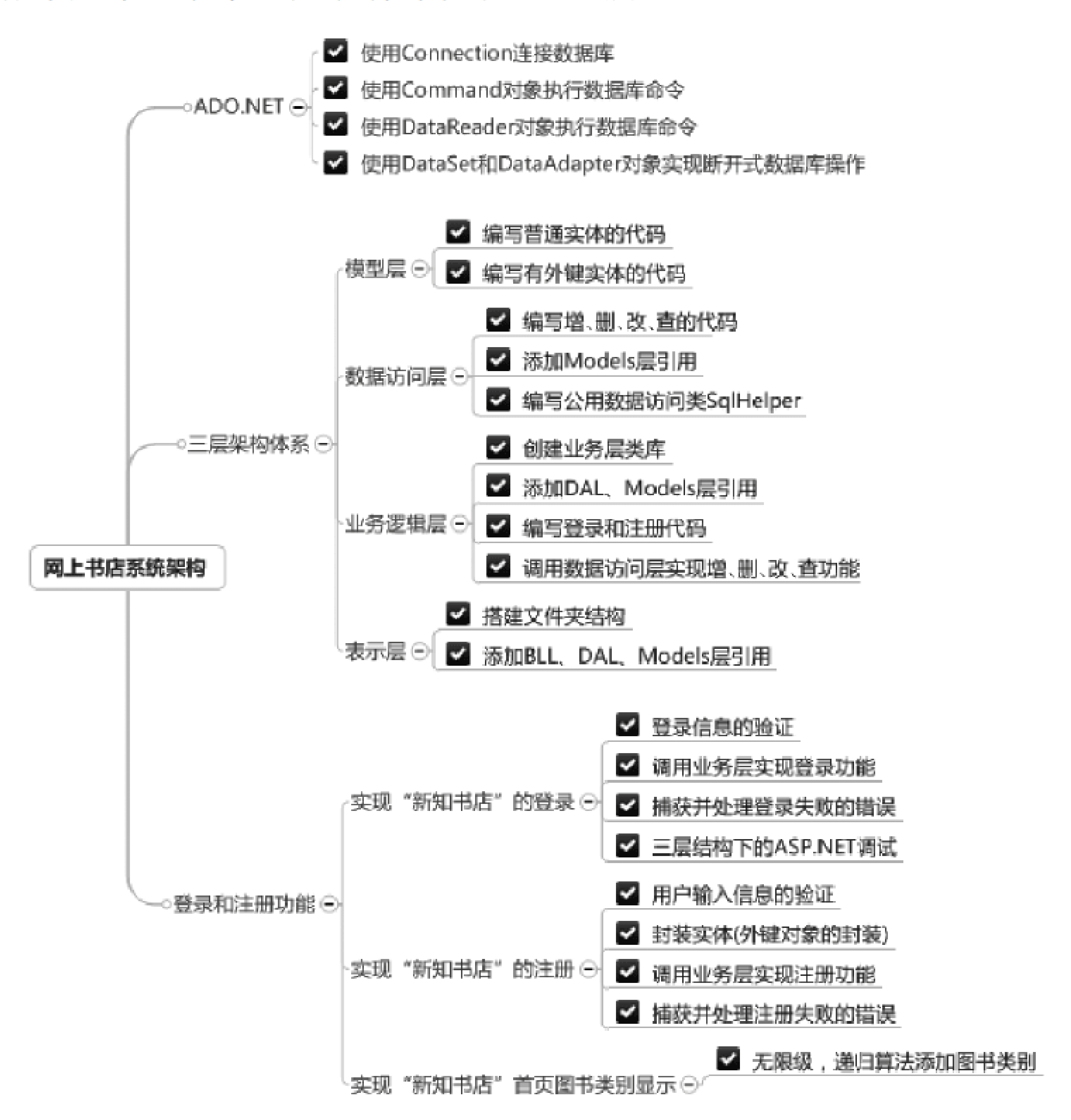


图 6-28　数据库访问及网上书店系统架构知识体系

6.5 单元练习题

一、选择题

1. (　　)对象用于从数据库中获取仅向前的只读数据流，并且在内存一次只能存放一行数据。此对象具有较好的功能，可以简单地读取数据。

A. DataAdapter　　B. DataSet　　C. DataView　　D. DataReader

2. Command 对象执行查询语句时，调用(　　)方法会返回结果集中的第一条记录的第一个字段的值。

A. ExecuteNoQuery　　B. ExecuteReader

C. ExecuteScalar　　D. ExecuteXmlReader

3. 如果要从数据库中获取多行记录，应该使用 Command 对象的(　　)方法。

A. ExecuteNoQuery　　B. ExecuteReader

C. ExecuteScalar

4. 如果要对数据库执行修改、插入和删除操作，应该使用 Command 对象的(　　)方法。

A. ExecuteNoQuery　　B. ExecuteReader

C. ExecuteScalar

5. (　　)是开发人员要使用的第一个对象，要求用于任何其他 ADO.NET 对象之前。

A. CommandBuilder 对象　　B. 命令对象

C. 连接对象　　D. DataAdapter 对象

6. (　　)表示一组相关表，在应用程序中这些表作为一个单元被引用。使用此对象可以快速从每一个表中获取所需的数据，当服务器断开时检查并修改数据，然后在下一次操作中就使用这些修改的数据更新服务器。

A. DataTable 对象　　B. DataRow 对象

C. DataReader 对象　　D. DataSet 对象

7. 数据适配器 DataAdapter 填充数据集的方法是(　　)。

A. Fill　　B. GetChanges　　C. AcceptChanges　　D. Update

8. 如果 Command 对象执行的是存储过程，其属性 CommandType 应取(　　)。

A. CommandType. Text　　B. CommandType. StoredProcedure

C. CommandType. TableDirect　　D. 没有限制

9. 如果希望将 FlightNumber 字段的值在包含信息字段的表的第一个<td>元素中显示，你要在表格的<td>元素添加(　　)代码以显示 FlightNumber 字段。

A. <td><%=FlightNumber%></td>

B. <td><script runat="server">FlightNumber</script></td>

C. <td><script>document. write("FlightNumber");</scripts></td>

D. <td>=FlightNumber</td>

10. 现在需要在使用三层架构搭建的某网上专卖店的网站上增加一个满 1000 送 200 的促销方案，在(　　)实现是最佳方式。

A. 模型层　　B. 表示层　　C. 数据访问层　　D. 业务逻辑层

11. 在ASP.NET中,若创建一个用户登录页面,使用用户表Users(Name,PassWord)中的数据,要求使用三层架构实现,下列说法(　　)是错误的。

A. 模型层通常包含与Users表相对应的实体类

B. 数据访问层封装了与Users表相关的增、删、改、查的操作

C. 判断输入账号是否合法的方法必须在表示层实现

D. 表示层负责内容的展示和与用户的交互

12. 下列说法不正确的是(　　)。

A. 数据访问层需要添加模型层的引用

B. 业务逻辑层需要添加数据访问层的应用

C. 表示层需要添加数据访问层、业务逻辑层和模型层的引用

D. 模型层需要添加数据访问层的引用

二、问答题

1. 列举常见的数据提供者,并且简单介绍对应的命名空间及作用。

2. 分别说明SqlCommand对象的ExecuteReader()、ExecuteNoQuery()和ExecuteScalar()方法的作用。

3. 简述DataSet与DataTable的区别与联系。

4. 简述SqlDataAdapter对象查询数据库数据的步骤。

5. 使用分层架构应遵守哪些原则?

单元7 数据绑定技术

教学目标：

- 了解数据绑定的类型、特性等相关概念。
- 掌握数据绑定的不同方法。
- 掌握数据源控件的使用。
- 掌握常用控件的数据绑定。

7.1 知识准备

7.1.1 数据绑定

在 ASP.NET 中，服务器控件可以直接与数据源进行交互（如显示或修改数据），ASP.NET 称这种技术为数据绑定技术。它可以把 Web 窗体页（包括其控件或其他元素）和数据源无缝地连接到一起，增强页与数据源的交互能力。数据绑定表达式的语法格式为：

```
<%#数据源%>
```

数据绑定允许在控件的声明代码中为控件的某个属性指定一个绑定表达式，从而将表达式的内容与该控件进行绑定。根据数据源的不同，数据绑定技术可以分为简单数据绑定技术和复杂数据绑定技术。

1. 简单数据绑定技术

简单数据绑定一般只绑定单个值到某个控件，所以数据源可以是表达式、变量、方法、控件的属性等。

（1）当绑定到 Label、TextBox 等控件时，需要将绑定表达式赋值给控件的 Text 属性：

```
Text = '<%#数据源%>'
```

（2）采用数据绑定技术还可以使用 JavaScript 调用 C#定义的变量和方法，此时可以将绑定表达式赋值给一个 JavaScript 变量：

```
var a = '<%#数据源%>'
```

【示例 7-1】　简单数据绑定的应用。

(1) 创建一个名字为 ch07 的站点,在站点中创建文件 SimpleDataBindDemo.aspx。

(2) 切换到页面 SimpleDataBindDemo.aspx 的设计视图,拖放 4 个 Label 控件至页面,切换到源视图,编写 JavaScript 代码,并分别为 4 个 Label 控件的 Text 属性编写数据绑定表达式,代码如表 7-1 所示。

表 7-1　页面 ConnectionDemo.aspx 的代码

行号	代　码　页
01	<head id = "Head1" runat = "server">
02	<title>简单绑定</title>
03	<script type = "text/javascript">
04	var a = '<%#str%>';
05	var b = '<%#func()%>';
06	document.write("数据绑定技术可以使用 javascript 调用" + a);
07	document.write("以及" + b);
08	</script>
09	</head>
10	<body>
11	<form id = "form1" runat = "server">
12	可以绑定表达式,显示当前时间为:<asp:Label ID = "Label1" runat = "server"
13	Text = '<%#DateTime.Now.ToString()%>'></asp:Label>
14	可以绑定:<asp:Label ID = "Label2" runat = "server" Text = '<%#str%>'></asp:Label>
15	可以绑定:<asp:Label ID = "Label3" runat = "server" Text = '<%#func()%>'></asp:Label>
16	可以绑定控件的属性值:<asp:Label ID = "Label4" runat = "server"
17	Text = '<%#Label1.Text%>'></asp:Label>
18	</form>
19	</body>
20	</html>

(3) 在 SimpleDataBindDemo.aspx.cs 文件中编写如下代码:

```
public partial class _Default : System.Web.UI.Page
{
    public string str = "C#语言定义的变量";
    protected void Page_Load(object sender, EventArgs e)
    {
        Page.DataBind();
    }
    public string func()
    {
        return "C#语言定义的方法";
    }
}
```

(4) 浏览该页面,结果如图 7-1 所示。

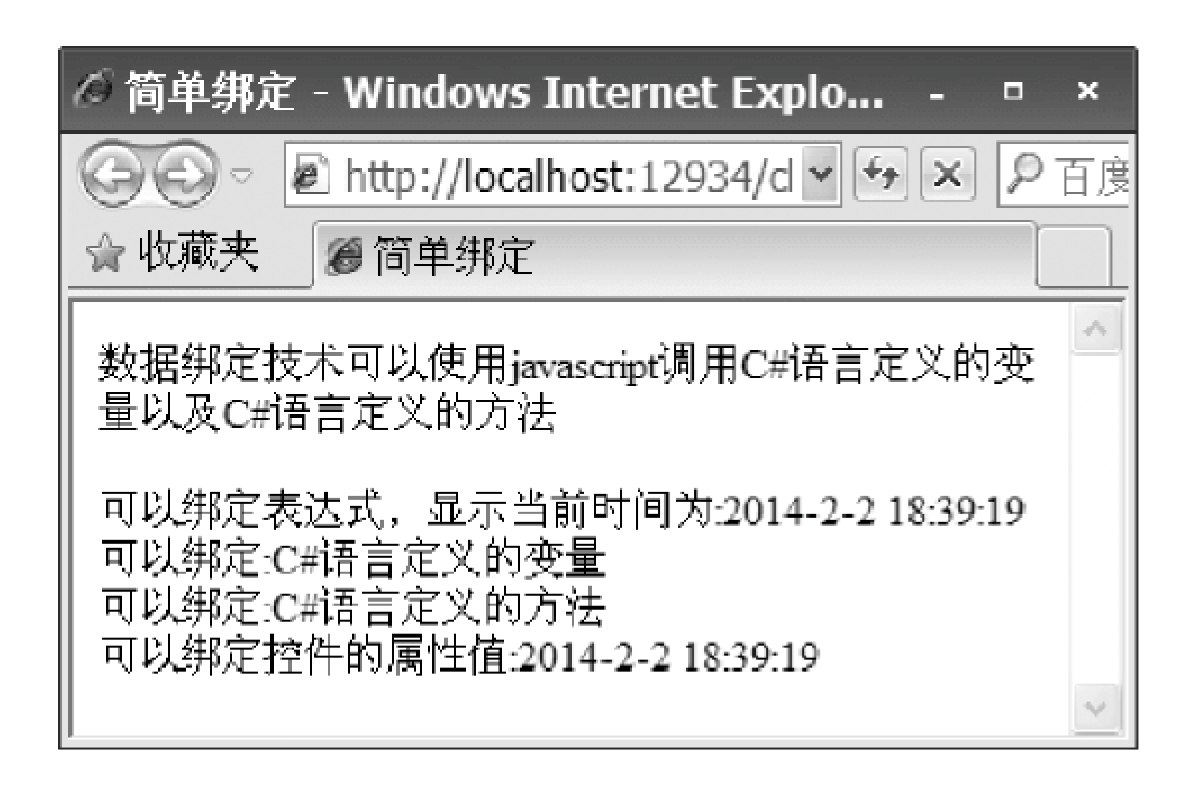


图 7-1　简单数据绑定示例

简单数据绑定需要注意以下几点:

(1) 数据绑定表达式不会自动计算它的值,除非它所在的页或者控件显式地调用了DataBind()方法,DataBind()方法能够将数据源绑定到被调用的服务器控件及其所有子控件。DataBind()是 Page 和所有服务器控件的方法,通常在 Page_Load 事件中被调用。可将案例 7-1 中的"Page. DataBind();"语句注释掉,再看一下运行结果。

(2) 绑定变量和方法的返回值时,该变量和方法必须声明为 public 或 protected 类型,否则会提示错误:"×××不可访问",因为它受保护级别限制。

(3) 如果数据绑定表达式中使用了双引号,则<%#数据源%>的最外层要用单引号,否则会提示"服务器标记的格式不正确"的错误信息,其他情况下使用双引号或者单引号都可以。将下面语句的单引号改成双引号即可得到印证。

```
<asp:Label ID = "Label1" runat = "server" Text = '<% # "单引号还是双引号?" %>'></asp:Label>
```

2. 复杂的数据绑定技术

复杂绑定就是将多个值绑定到数据绑定控件的某个属性上。拥有多个值的数据源有集合、DataTable、DataSet 等,后续章节将分别介绍。

复杂绑定时,需要在前台将绑定表达式赋值给控件的 DataSource 属性:

```
DataSource = '<% # 数据源 %>'
```

或者在后台将数据源赋值给控件的 DataSource 属性:

```
控件名.DataSource = 数据源
```

【示例 7-2】 复杂数据绑定的应用。

(1) 在站点 ch07 中创建页面文件 ComplexBindDemo. aspx。

(2) 切换到页面 ComplexBindDemo. aspx 的设计视图,拖放 1 个 ListBox 控件至页面,切换到源视图,编写如下所示代码:

```
<body>
  <form id="form1" runat="server">
    <asp:ListBox ID="lbColor" runat="server" Rows="6" Width="200px" AutoPostBack=
"True"
                  OnSelectedIndexChanged="lbColor_SelectedIndexChanged">
    </asp:ListBox>
  </form>
</body>
```

(3) 编写 ComplexBindDemo.aspx.cs 文件中代码如表 7-2 所示。

表 7-2 页面 ComplexBindDemo.aspx 的后台 cs 文件代码

```
protected void Page_Load(object sender, EventArgs e)
{
    ArrayList cls = new ArrayList();
    if (!IsPostBack)
    {
        //创建颜色数组
        cls.Add("Red");
        cls.Add("Blue");
        cls.Add("Green");
        cls.Add("Black");
        cls.Add("Yellow");
        cls.Add("Gray");
        //把 cls 设置为 ListBox 控件 lbColor 的数据源,并绑定控件的数据
        lbColor.DataSource = cls;
        lbColor.DataBind();
    }
}

protected void lbColor_SelectedIndexChanged(object sender, EventArgs e)
{
    if (lbColor.SelectedIndex > -1)
    { //把控件的前景颜色设置为选择项的颜色
        lbColor.ForeColor = Color.FromName(lbColor.SelectedItem.Text);
    }
}
```

(4) 浏览该页面,结果如图 7-2 所示。

本示例创建了一个名为 cls 的 ArrayList 对象,并将其作为 ListBox 控件对象 lbColor 的数据源进行绑定,lbColor 还定义了 SelectedIndexChanged 事件,该事件把 lbColor 控件的前景颜色设置为当前选择项所指定的颜色。

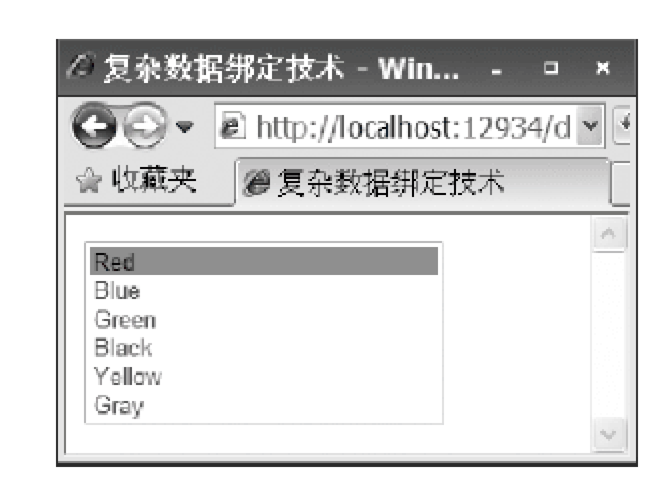


图 7-2 复杂数据绑定示例

3. Eval()和 Bind()方法

Eval()和 Bind()方法是数据绑定的两种重要方法。

（1）Eval()方法：取属性的名称为参数，并返回其内容。它仅用于只读的单向数据绑定情况。它实现了数据读取的自动化，但是没有实现数据写入自动化。其语法如下：

```
<%# Eval(属性名称) %>
```

例如：

```
<asp:Label ID="st_idLabel" runat=server Text=<%#Eval("st_id")%>/>
```

上述代码将 st_id 字段的值绑定到 Label 控件 st_idLabel 的 Text 属性上。

```
发布时间:<%#Eval("DateTime","{0:yyyy-mm-dd,hh:mm:ss}")%>
```

上述代码将 DateTime 字段的值以“年-月-日，时:分:秒”的格式呈现在浏览器上。

（2）Bind()方法：Bind()方法支持双向数据绑定——既能把数据绑定到控件，又能把数据变更提交到数据库。它实现了数据读取的自动化，也实现了数据写入自动化。语法与 Eval 方法的语法类似：

```
<%# Bind(属性名称) %>/>
```

例如：

```
<asp:TextBox ID="st_nameTextBox" runat="server" Text='<%# Bind("st_name") %>'/>
```

上述代码将 st_name 字段的值绑定到 TextBox 控件 st_nameTextBox 的 Text 属性上。

4. 数据绑定方式

ASP.NET 提供了以下两种数据绑定方式。

（1）编码指定数据源：就是编写代码在程序运行中动态绑定数据源，比如：

```
this.gvMain.DataSource = new UserManager().GetNormalUsers();
this.gvMain.DataBind();
```

其中，gvMain 是数据绑定控件 GridView，业务逻辑层的 GetNormalUsers()方法返回的是类型为 List＜User＞的所有正常用户数据。

（2）使用数据源控件：数据源控件用于实现从不同数据源（数据库、XML 文件或业务逻辑层对象）获取数据的功能，它可以设置连接信息、查询信息、参数和行为，这样就可以把指定的数据绑定到数据绑定控件上。

7.1.2 数据源控件

ASP.NET 包含一些数据源控件，这些数据源控件允许连接不同类型的数据源，这些数据源包括 ADO.NET 容器类（DataSet、DataTable、DataView）、数据读取器（DataReader）、中间层业务对象及自定义的集合、字典和数组等。

其实，数据源控件我们在站点导航中已经使用过，SiteMapDataSource 控件就是数据源控件。

所有数据源控件都是由 System. Web. UI. DataSourceControl 类派生而来，图 7-3 是数据源控件类的层次结构图。

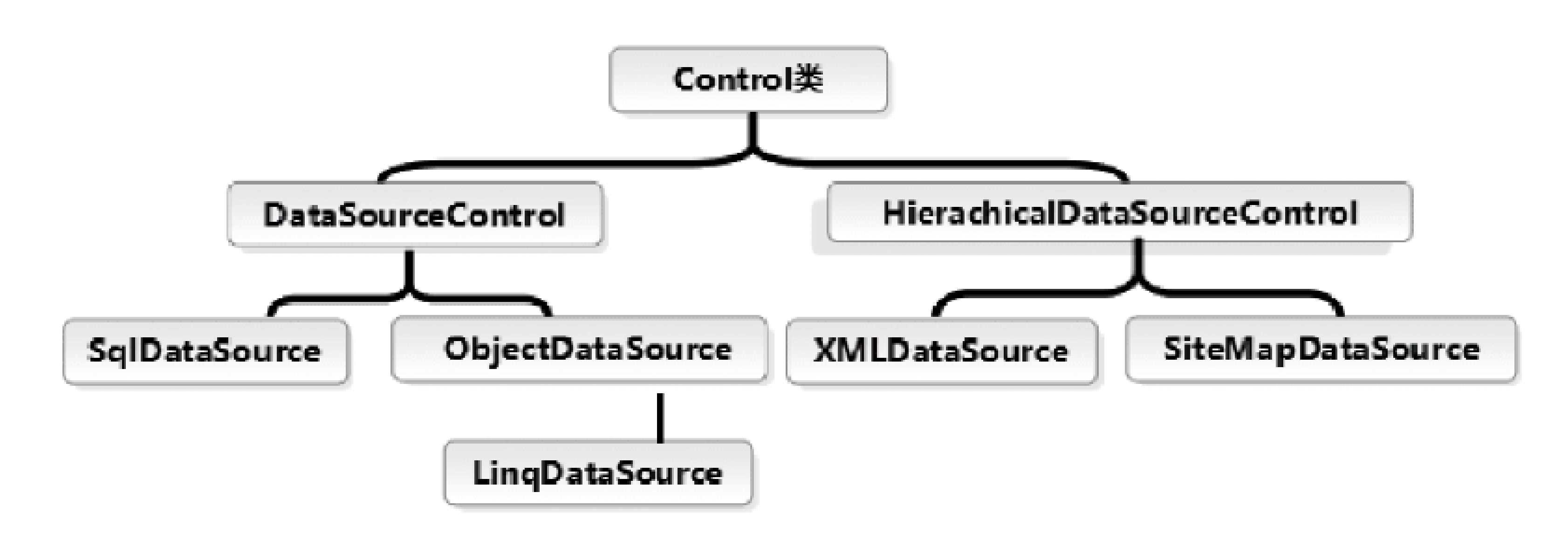


图 7-3　数据源控件类的层次结构图

图 7-3 中的普通数据源控件就是 DataSourceControl 控件，层次化数据源控件就是 HierachicalDataSourceControl。表 7-3 列出了 ASP.NET 4.0 中的 6 个数据源控件及其作用，在 Visual Studio 的工具箱中的"数据"标签中可以找到。

表 7-3　ASP.NET 4.0 内置的数据源控件

数据源控件	描　　述
SqlDataSource	支持绑定到 ADO.NET 提供程序表示的 SQL 数据库。与 SQL Server 一起使用时支持高级缓存功能。当数据作为 DataSet 对象返回时，此控件支持排序、筛选和分页
AccessDataSource	支持绑定到 Microsoft Access 数据库。当数据作为 DataSet 对象返回时，此控件支持排序、筛选和分页
ObjectDataSource	支持绑定到业务对象或其他类以及创建依赖中间层对象（业务逻辑层）管理数据的 Web 应用程序。支持对其他数据源控件不可用的高级排序和分页方案
SiteMapDataSource	专门处理类似站点地图的 XML 数据。默认情况下，数据源是以. sitemap 为扩展名的 XML 文件
XmlDataSource	用于检索和处理 XML 等分层数据。它可以从文件、URL 或者包含 XML 内容的字符串中加载 XML 数据
LinqDataSource	使用语言集成查询(LINQ)从数据对象中检索和修改数据

这些数据源控件将在后续章节和数据控件结合进行举例说明

1. SqlDataSource 数据源控件

SqlDataSource 控件用于连接到 SQL 关系数据库的数据源。其中包括 Microsoft SQL Server 和 Oracle 数据库以及 OLE DB 和 ODBC 数据源。将 SqlDataSource 控件与数据绑定控件一起使用，可以从关系数据库中检索数据、在 ASP.NET 网页上显示和操作数据。该控件提供了一个易于使用的向导，可引导用户完成配置过程，也可以通过直接修改控件的属性，手动修改控件，不必编写代码或只需编写少量代码。表 7-4 列出了为 SqlDataSource 控件支持的数据库操作属性。

表 7-4 SqlDataSource 控件的主要属性

属 性	说 明
DeleteCommand	获取或设置 SqlDataSource 控件删除数据库数据所用的 SQL 命令
DeleteCommandType	获取或设置删除命令类型,可取的值为 Text、StoredProduce,分别对应 SQL 命令、存储过程
DeleteParameters	获取 DeleteCommand 属性所使用的参数的集合
InsertCommand	获取或设置 SqlDataSource 控件插入数据库数据所用的 SQL 命令
InsertCommandType	获取或设置插入命令类型,可取的值为 Text 和 StoredProduce
InsertParameters	获取 InsertCommand 属性所使用的参数的参数集合
SelectCommand	获取或设置 SqlDataSource 控件查询数据库数据所用的 SQL 命令
SelectCommandType	获取或设置查询命令类型,可取的值为 Text 和 StoredProduce
SelectParameters	获取 SelectCommand 属性所使用的参数的参数集合
UpdateCommand	获取或设置 SqlDataSource 控件更新数据库数据所用的 SQL 命令
UpdateCommandType	获取或设置更新命令类型,可取的值为 Text 和 StoredProduce
UpdateParameters	获取 UpdateCommand 属性所使用的参数的参数集合
DataSourceMode	表明在 SqlDataSource 控件检索数据时,是使用 DataSet 还是使用 DataReader
ProviderName	获取或设置.NET Framework 数据提供程序的名称

2. ObjectDataSource 数据源控件

大多数 ASP.NET 数据源控件,如 SqlDataSource 都是在两层应用程序层次结构中使用。在该层次结构中,表示层(ASP.NET 网页)可以与数据层(数据库和 XML 文件等)直接进行通信。但是,常用的应用程序设计原则是将表示层与业务逻辑相分离,而将业务逻辑封装在业务逻辑层(BLL)中。这些业务对象在表示层(UI)和数据访问层(DAL)之间形成一层,从而生成一种 3 层应用程序结构。

ObjectDataSource 控件通过提供一种将相关页上的数据控件绑定到中间层业务对象的方法,为 3 层结构提供支持。在不使用扩展代码的情况下,ObjectDataSource 控件使用中间层业务对象以声明方式对数据执行选择、插入、更新、删除、分页、排序、缓存和筛选操作。ObjectDataSource 控件的主要属性如表 7-5 所示。

表 7-5 ObjectDataSource 控件的主要属性

属 性	说 明
DelectMethod	获取或设置由 ObjectDataSource 控件调用以删除数据的方法或函数的名称
DeleteParameters	获取或设置参数集合,该集合包含由 DeleteMethod 方法使用的参数
InsertMethod	获取或设置由 ObjectDataSource 控件调用以插入数据的方法或函数的名称
InsertParameters	获取或设置参数集合,该集合包含由 InsertMethod 方法使用的参数
SelectMethod	获取或设置由 ObjectDataSource 控件调用以查询数据的方法或函数的名称
SelectParameters	获取或设置参数集合,该集合包含由 SelectMethod 方法使用的参数
UpdateMethod	获取或设置由 ObjectDataSource 控件调用以更新数据的方法或函数的名称
UpdateParameters	获取或设置参数集合,该集合包含由 UpdateMethod 方法使用的参数
FilterExpression	获取或设置当调用由 SelectMethod 属性指定的方法时应用的筛选表达式
FilterParameters	获取或设置与 FilterExpression 字符串中的任何参数占位符关联的参数的集合

续表

属　性	说　明
EnableCaching	获取或设置一个值，该值指示 ObjectDataSource 控件是否启用数据缓存
SelectCountMethod	获取或设置由 ObjectDataSource 控件调用以检索行数的方法或函数的名称
TypeName	获取或设置 ObjectDataSource 控件要调用的类的名称

3. SiteMapDataSource 数据源控件

SiteMapDataSource 控件用于 ASP.NET 站点导航。该控件检索站点地图提供程序的导航数据，并将该数据传递到可显示该数据的控件。

站点地图是表示一个 Web 站点中存在的所有页面和目录的图，用来向用户展示他们正在访问的页面的逻辑坐标，允许用户动态地访问站点位置，并以图形方式生成所有的导航数据。导航数据包括有关网站中的页的信息，如 URL、标题、说明和导航层次结构中的位置。若将导航数据存储在一个地方，则可以更方便地在网站的导航菜单中添加和删除项。由于站点地图是一种层次性信息，将 SiteMapDataSource 控件的输出绑定到层次性数据绑定控件(诸如 TreeView)，使它能够显示站点的结构。

站点地图信息可以以很多种形式出现，其中最简单的形式是位于应用程序的根目录中的一个名为 web.sitemap 的 XML 文件。SiteMapDataSource 控件可以处理存储在 Web 站点的 SiteMap 配置文件中的数据。

7.1.3　常用控件的数据绑定

下面就以前面章节中介绍过的 RadioButtonList、CheckBoxList 和 DropDownList 控件为例来了解数据绑定。

1. RadioButtonList 控件的数据绑定

使用 RadioButtonList 控件进行数据绑定之前，先了解一下它的相关属性和事件。RadioButtonList 控件的常用属性和事件如表 7-6 所示。

表 7-6　RadioButtonList 控件的常用属性和事件

属性和事件		说　明
属性	AutoPostBack	指示当用户更改列表中的选定内容时是否自动产生向服务器的回传
	DataTextField	为列表项提供文本内容的数据源字段
	DataValueField	为各列表项提供值的数据源字段
	SelectedIndex	选定项的索引
	SelectedItem	获取列表控件中的选定项
	SelectedValue	获取列表控件中选定项的值
事件	SelectedIndexChanged	当列表控件的选定项在信息发往服务器之间变化时触发

除 SelectedValue 之外，通过 SelectedItem. Text 和 SelectedItem. Value 可获得选择项的文本内容和值。

【示例 7-3】　实现 RadioButtonList 控件绑定到数据库，当选择 RadioButtonList 中的

某个学生时,实时输出该学生的详细信息。

(1) 在站点 ch07 中创建页面文件 RadioButtonListDemo.aspx。

(2) 切换到页面 RadioButtonListDemo.aspx 的设计视图,拖放 1 个 RadioButtonList 控件至页面,切换到源视图,编写如下代码:

```
<form id = "form1" runat = "server">
    <div>
        <asp:RadioButtonList ID = "rblStuInfo" runat = "server"
            OnSelectedIndexChanged = "rblStuInfo_SelectedIndexChanged"
            RepeatDirection = "Horizontal" AutoPostBack = "True">
        </asp:RadioButtonList>
    </div>
</form>
```

(3) 打开 RadioButtonListDemo.aspx.cs 文件,编写一个方法,代码如表 7-7 所示。GetTable()方法用于连接数据库,并根据命令字符串 Commstr 结合 DataAdpter 和 DataSet 构建 DataTable 对象并返回。

表 7-7 方法 GetTable()的代码

```
/// <summary>
/// 根据 SQL 查询语句构建 DataTable 对象
/// </summary>
/// <param name = "Commstr"> SQL 查询字符串</param>
/// <returns></returns>
private DataTable GetTable(string Commstr)
{
    DataSet ds = new DataSet(); //建立 DataSet 对象
    string sqlstr = WebConfigurationManager.ConnectionStrings["StuConnString"].ConnectionString;
    SqlConnection sqlconn = new SqlConnection(sqlstr);
    sqlconn.Open(); //打开连接
    SqlDataAdapter sda = new SqlDataAdapter(Commstr, sqlconn); //建立 DataAdapter 对象
    sda.Fill(ds, "StuTable");//用 Fill 方法返回的数据,填充 DataSet,数据表取名为"StuTable"
    DataTable dtable = ds.Tables["StuTable"];  //将数据表 StuTable 的数据复制到
                                                //DataTable 对象
    return dtable;
}
```

(4) 在 RadioButtonListDemo.aspx.cs 文件中,为 Page_Load 事件方法编写代码如下:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
```

```
            string cmdStr = "select Name from StuInfo";
            rblStuInfo.DataSource = GetTable(cmdStr).DefaultView;
            rblStuInfo.DataTextField = "Name";
            rblStuInfo.DataBind();
        }
    }
```

此时，运行页面 RadioButtonListDemo.aspx，结果如图 7-4 所示，单击任何单选按钮均无对应信息出现。



图 7-4　Name 字段绑定到 RadioButtonList 控件

（5）在 RadioButtonListDemo.aspx.cs 文件中，编写 RadioButtonList 控件的 SelectedIndexChanged 事件方法代码，同时编写一个显示选定学生详细信息的 ShowStuInfo 方法，如表 7-8 和表 7-9 所示。

表 7-8　方法 ShowStuInfo()的代码

行号	代　码　页

```
/// <summary>
/// 根据 rblStuInfo 控件选择项显示学生信息
/// </summary>
/// <param name = "dtable"> DataTable 对象</param>
private void ShowStuInfo(DataTable dtable)
{
    DataRowCollection drc = dtable.Rows;       //用 DataRowCollection 对象获取 StuTable
                                               //数据表的所有数据行
    DataRow dr;                                //建立 DataRow 数据行对象
    for (int i = 0; i < drc.Count; i++)        //逐行遍历,取出各行的数据
    {
        dr = drc[i];
        Response.Write("学号:" + dr["StuNo"] + "姓名:" + dr["Name"] + "性别:" + dr["Sex"] +
                       " 出生日期: " + dr["Birth"] + " 照片: " + dr["Photo"]);
        Response.Write("<br />");
    }
}
```

此时，运行页面 RadioButtonListDemo.aspx，选中任何单选按钮，将出现与该姓名对应的详细信息，如图 7-5 所示。

表 7-9　SelectedIndexChanged()事件方法的代码

```
/// <summary>
/// rblStuInfo 控件的 SelectedIndexChanged 事件方法
/// </summary>
/// <param name = "sender"></param>
/// <param name = "e"></param>
protected void rblStuInfo_SelectedIndexChanged(object sender, EventArgs e)
{
    string str = rblStuInfo.SelectedValue;
    string cmdstr = "select * from StuInfo where Name = '" + str + "'";
    DataTable dt = GetTable(cmdstr);
    ShowStuInfo(dt);
}
```



图 7-5　RadioButtonList 控件的数据绑定

注意：运行时若选择一个姓名后无对应信息显示，可能有两个原因：一是没有将 RadioButtonList 控件的 AutoPostBack 属性设置为 True；二是 Page_Load 事件方法代码没有用 if(! IsPostBack)语句判断是否为第一次加载。

2. DropDownList 控件的数据绑定

DropDownList 下拉列表控件的常用属性和事件如表 7-10 所示。

表 7-10　DropDownList 控件的常用属性和事件

属性和事件		说　明
属性	AutoPostBack	指示当用户更改列表中的选定内容时是否自动产生向服务器的回传
	DataTextField	为列表项提供文本内容的数据源字段
	DataValueField	为各列表项提供值的数据源字段
	SelectedIndex	选定项的索引
	SelectedItem	获取列表控件中的选定项
	SelectedValue	获取列表控件中选定项的值
事件	SelectedIndexChanged	当列表控件的选定项在信息发往服务器之间变化时触发

除 SelectedValue 之外，通过 SelectedItem. Text 和 SelectedItem. Value 可获得选择项的文本内容和值。对比 RadioButtonList 控件的常用属性，发现没有区别。

【示例 7-4】 用 DropDownList 控件绑定到数据库，模拟用户注册时省市的选择，实现省份城市级联效果。所用数据库的数据表结构如图 7-6 所示。

表 - dbo.province　摘要

ProovinceID	Name
1	北京市
2	天津市
3	上海市
4	重庆市
5	河北省
6	山西省
7	台湾地区

表 - dbo.city　表 - dbo.province

ID	CityName	PID
1	北京市	1
5	本溪市	8
6	毕节地区	23
16	滨州市	16
15	亳州市	13
17	博乐市	31
9	沧州市	5

图 7-6　Province 和 City 表结构

(1) 在 web. config 中添加数据库连接字符串，名为 ProvinceConnString，用来连接到数据库 Province。

(2) 右击站点项目 ch07，在弹出的快捷菜单中选择“添加 ASP.NET 文件夹”→App_Code 命令，添加 App_Code 文件夹。

在 App_Code 文件夹中编写实体类 City. cs 和 Province. cs，代码如下。

```
public class City
{
    public int CityID { get; set; }                          //城市 ID
    public string CityName { get; set; }                     //城市名称
    public Province pro;                                     //城市所在省份 ID
}

public class Province
{
    public int ProvinceID{get;set;}                          //省份 ID
    public string ProvinceName{get;set;}                     //省份名称
}
```

在 App_Code 文件夹中编写数据库操作类 SqlOperator. cs，代码如表 7-11 所示。

表 7-11　数据库操作类 SqlOperator. cs 的代码

行号	代　码　页
01	public abstract class SqlOperator
02	{
03	//数据库连接字符串
04	public static readonly string ConnectionString =
05	ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString;
06	/// <summary>
07	/// 获取所有省份名称

续表

代 码 页

```
    /// </summary>
    /// <returns></returns>
    public static List<Province> GetProvince()
    {
        string sql = "Select [provinceId],name from province";
        List<Province> provinceList = new List<Province>();
        SqlDataReader dr = GetDataReader(sql);
        while (dr.Read())
        {
            Province pro = new Province();
            pro.ProvinceID = Convert.ToInt32(dr["ProvinceID"]);
            pro.ProvinceName = dr["name"].ToString();
            provinceList.Add(pro);
        }
        dr.Close();
        return provinceList;
    }
    /// <summary>
    /// 根据省份 ID 号获取城市列表
    /// </summary>
    /// <param name="pid">省份 ID</param>
    /// <returns></returns>
    public static List<City> GetCityByProvinceID(string pid)
    {
        List<City> cityList = new List<City>();
        string sqlStr = string.Format("Select [ID],[CityName] from CITY  WHERE PID = {0}", pid);
        SqlDataReader dr = GetDataReader(sqlStr);
        while (dr.Read())
        {
            City city = new City();
            city.CityID = Convert.ToInt32(dr["ID"]);
            city.CityName = dr["CityName"].ToString();
            cityList.Add(city);
        }
        dr.Close();
        return cityList;
    }
    /// <summary>
    /// 根据 SQL 语句创建 SqlDataReader 数据集
    /// </summary>
    /// <param name="sqlStr"> SQL 查询语句</param>
```

续表

```
        /// <returns></returns>
        private static SqlDataReader GetDataReader(string sqlStr)
        {
            SqlDataReader dr = null;
            SqlCommand cmd  =  new SqlCommand();
            SqlConnection conn  =  new SqlConnection(ConnectionString);
            try
            {
                conn.Open();
                cmd.Connection  =  conn;
                cmd.CommandType  =  CommandType.Text;
                cmd.CommandText  =  sqlStr;
                dr  =  cmd.ExecuteReader(CommandBehavior.CloseConnection);
                return dr;
            }
            catch
            {
                conn.Close();
                throw;
            }
        }
    }
```

（3）右击站点项目 ch07 下的文件夹 ch7_4，通过快捷菜单命令新建一个名为 DropDownListBindDemo.aspx 的页面，切换到页面的设计视图，拖放 2 个 DropDownList 控件至页面，切换到源视图，编写如下所示代码：

```
<body>
  <form id="form1" runat="server">
        <div>
            <asp:DropDownList ID="ddlProvince" runat="server" AutoPostBack="True"
                 OnSelectedIndexChanged="ddlProvince_SelectedIndexChanged">
            </asp:DropDownList>
            省
            <asp:DropDownList ID="ddlCities" runat="server">
            </asp:DropDownList>
            市
        </div>
   </form>
</body>
```

（4）编写 DropDownListBindDemo.aspx 页面的 Page_Load 事件方法代码如表 7-12 所示。

表 7-12 页面 DropDownListBindDemo.aspx 的 Page_Load 事件方法代码

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.BindProvince();
        this.BindCities(this.ddlProvince.SelectedValue);
    }
}
```

其中，绑定省份和城市分别由方法 BindProvince()和方法 BindCities()实现，代码如表 7-13 和表 7-14 所示。

表 7-13 方法 BindProvince()的代码

```
/// <summary>
/// 绑定省份
/// </summary>
private void BindProvince()
{
    List<Province> pList = SqlOperator.GetProvince();
    this.ddlProvince.DataSource = pList;
    this.ddlProvince.DataTextField = "ProvinceName";
    this.ddlProvince.DataValueField = "ProvinceID";
    this.ddlProvince.DataBind();
    this.ddlProvince.Items.Insert(0, new ListItem("=== 请选择 ===", "0"));
}
```

表 7-14 方法 BindCities()的代码

```
    /// <summary>
    /// 绑定城市
    /// </summary>
    /// <param name="provinceId">省份 Id</param>
    private void BindCities(string provinceId)
    {
        ddlCities.Items.Clear();
        List<City> cities = SqlOperator.GetCityByProvinceID(provinceId);
        ddlCities.DataSource = cities;
        ddlCities.DataTextField = "CityName";
        ddlCities.DataValueField = "CityID";
        ddlCities.DataBind();
    }
```

(5) 当 ddlProvince 中选择了某省份时，需要将该省下属的城市名绑定到 ddlCities，在 DropDownListBindDemo.aspx.cs 文件中编写 DropDownList 控件对象 ddlProvince 的 SelectedIndexChanged 事件方法的代码如表 7-15 所示。

表 7-15　SelectedIndexChanged()事件方法的代码

```
protected void ddlProvince_SelectedIndexChanged(object sender, EventArgs e)
{
    if (this.ddlProvince.SelectedValue == "0")
    {
        return;
    }
    string provinceId = this.ddlProvince.SelectedValue;
    this.BindCities(provinceId);
}
```

此时，运行页面 DropDownListBindDemo.aspx，效果如图 7-7 所示，选择任何一个省份，均会绑定该省下属的城市到 ddlCities。

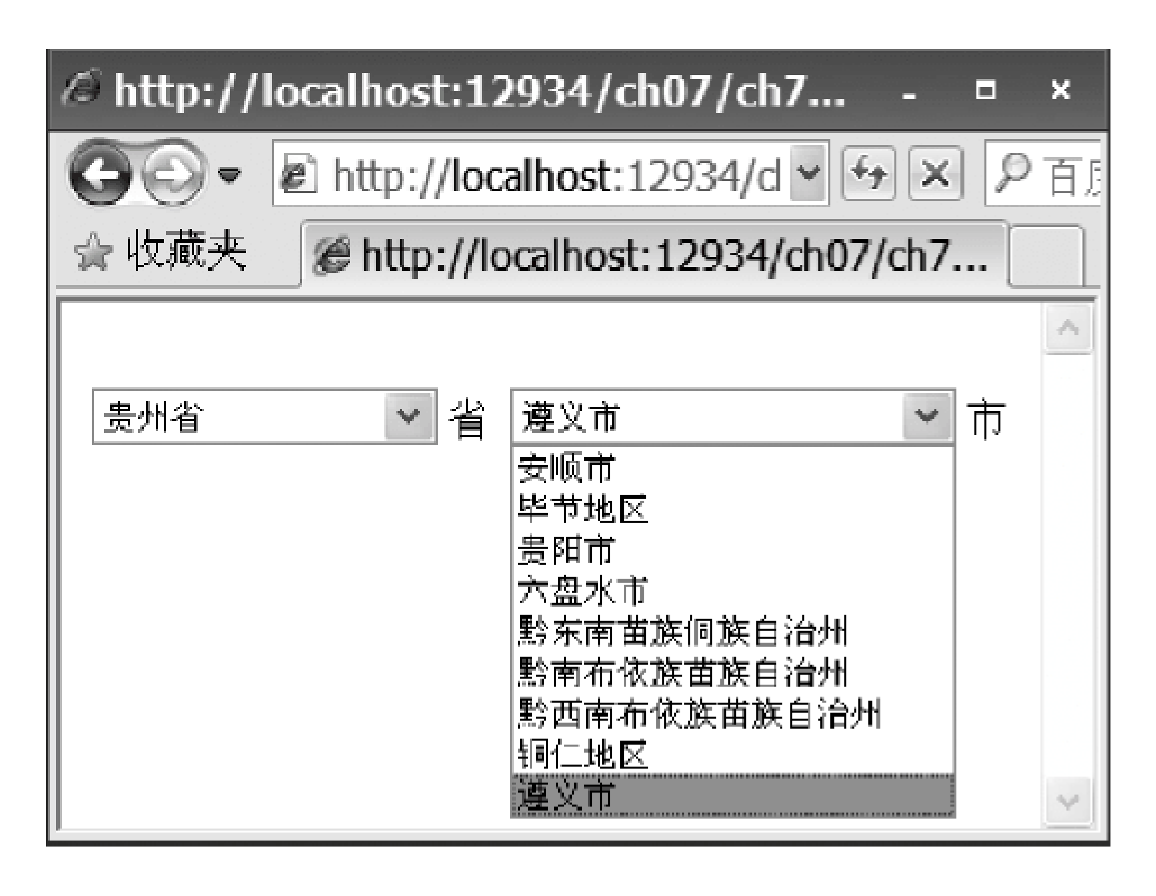


图 7-7　省份城市的级联效果

本示例虽然没有明确采用 3 层架构创建模型层(Model)、数据访问层(DAL)、业务逻辑层(BLL)及表示层(UI)，但逻辑上已经是 3 层架构，负责不同功能的程序代码分别写在不同的类文件中。

7.2　单元任务

任务 7-2-1　实现“新知书店”后台图书列表中的检索类别选择

【任务描述】

要实现“新知书店”图书按类别进行显示，就要具备检索类别的选择，使用

DropDownList 控件实现“新知书店”后台书籍列表页中检索类别的选择功能。所使用的检索类别数据表为第一单元阐述系统业务需求分析与设计时已经描述过的 Categories 表，要求实现如图 7-8 所示的将该表中的数据绑定到指定 DropDownList 中的效果。



图 7-8　书籍列表中的检索类别选择功能效果

【任务实施】

1. 图书检索类别选择数据访问层与业务逻辑层的实现

要实现图书检索类别的选择，仍需要从数据访问层、业务逻辑层和表示层分别进行编码。

1）图书检索类别选择数据访问层的实现

在 BookShopDAL 项目中新建类文件 SelectByCategoryService.cs，代码如下：

```
⋮
using BookShop.Models;
using System.Configuration;
namespace BookShop.DAL
{
  public class SelectByCategoryService
  {
    string connection = ConfigurationManager.ConnectionStrings["BookShop"].ConnectionString;
    /// <summary>
    /// 查询所有图书类别的所有字段信息
    /// </summary>
    /// <returns></returns>
    public List<SelectByCategory> GetSelectByCategory()
    {
        //使用 泛型集合 List<T>的方式传递实体对象集合
        List<SelectByCategory> SelectByCategoryList = new List<SelectByCategory>();
```

```
            string sql = "SELECT *  FROM Categories";
            using (SqlDataReader reader = SqlHelper.ExecuteReader(this.connection,
                                                       CommandType.Text, sql))
            {
                while(reader.Read())
                {
                    SelectByCategory selectbycategory = new SelectByCategory();
                    selectbycategory.Id = (int)reader["Id"];
                    selectbycategory.Name = (string)reader["Name"];
                    SelectByCategoryList.Add(selectbycategory);
                }
            }
            return SelectByCategoryList;
        }
    }
}
```

2）图书检索类别选择业务逻辑层的实现

在 BookShopBLL 项目中新建类文件 SelectByCategoryManager.cs，代码如下：

```
⋮
using BookShop.Models;
using BookShop.DAL;
namespace BookShop.BLL
{
    public class SelectByCategoryManager
    {
        public List<SelectByCategory> GetSelectByCategory()
        {
            return new SelectByCategoryService().GetSelectByCategory();
        }
    }
}
```

2. 图书检索类别选择表示层的实现

1）表示层页面设计

在 Web 站点项目的文件夹 Admin 下，根据后台母版页 Admin.master 新建图书列表内容页 BookList.aspx，并从工具箱拖入 DropDownList 控件至页面并修改其 ID 属性为 ddlQueryCategories，代码如下：

```
<asp:Content ID="Content1" ContentPlaceHolderID="cphAdmin" runat="Server">
    检索类别:
    <asp:DropDownList ID="ddlQueryCategories" runat="server" Height="16px">
    </asp:DropDownList>
</asp:Content>
```

2）编程实现图书类别数据绑定到 DropDownList 控件

在图书列表页的后置代码文件 BookList.aspx.cs 中编程实现将图书类别数据绑定到 DropDownList 控件的代码如下：

```
⋮
using BookShop.BLL;
using BookShop.Models;

public partial class Admin_BookList : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            BindSelectByCategory();
        }
    }

    protected void BindSelectByCategory()
    {
        this.ddlQueryCategories.DataSource = new SelectByCategoryManager().GetSelectByCategory();
        this.ddlQueryCategories.DataTextField = "Name";      //用于显示的字段
        this.ddlQueryCategories.DataValueField = "Id";       //用于存值的字段
        this.ddlQueryCategories.DataBind();
        this.ddlQueryCategories.Items.Insert(0, new ListItem("===请选择===", "0"));
    }
}
```

运行页面 BookList.aspx，效果如图 7-8 所示。在编程实现图书类别数据绑定到 DropDownList 控件上的时候，很多读者会将代码写成如下形式：

```
protected void Page_Load(object sender, EventArgs e)
{
     if (!IsPostBack)
     {
          this.ddlQueryCategories.DataSource = newSelectByCategoryManager().GetSelectByCategory();
          this.ddlQueryCategories.DataBind();
     }
}
```

这样运行程序将会出现图 7-9 所示的效果。

出现如图 7-9 所示错误的原因是 DropDownList 控件所绑定的是 GetSelectByCategory 对象而不是检索类别信息的名字，因此需要使用 DataTextField 属性为 DropDownList 设置显示字段，另外，可以使用 DataValueField 属性来设置实际存储的字段。



图 7-9　运行上述代码后的效果

7.3　项目实训

实现“博客系统”文章发表功能

【需求说明】

- 使用此功能的只能是注册会员。
- 对用户输入的数据进行非空验证。
- 用户输入文章的标题和内容，将文章信息添加到数据库，效果如图 7-10 所示。



图 7-10　“博客系统”文章发表页效果(PublishAticle.aspx)

(提示：使用 CKeditor 在线编辑控件完成编辑功能，避免纯文本文档的单调。)

7.4　单元小结

本单元介绍 ASP.NET 中的数据绑定技术，首先介绍简单绑定与复杂绑定技术，阐述数据绑定表达式，并通过实例讲解常用控件如何绑定数据，本单元知识体系如图 7-11 所示。

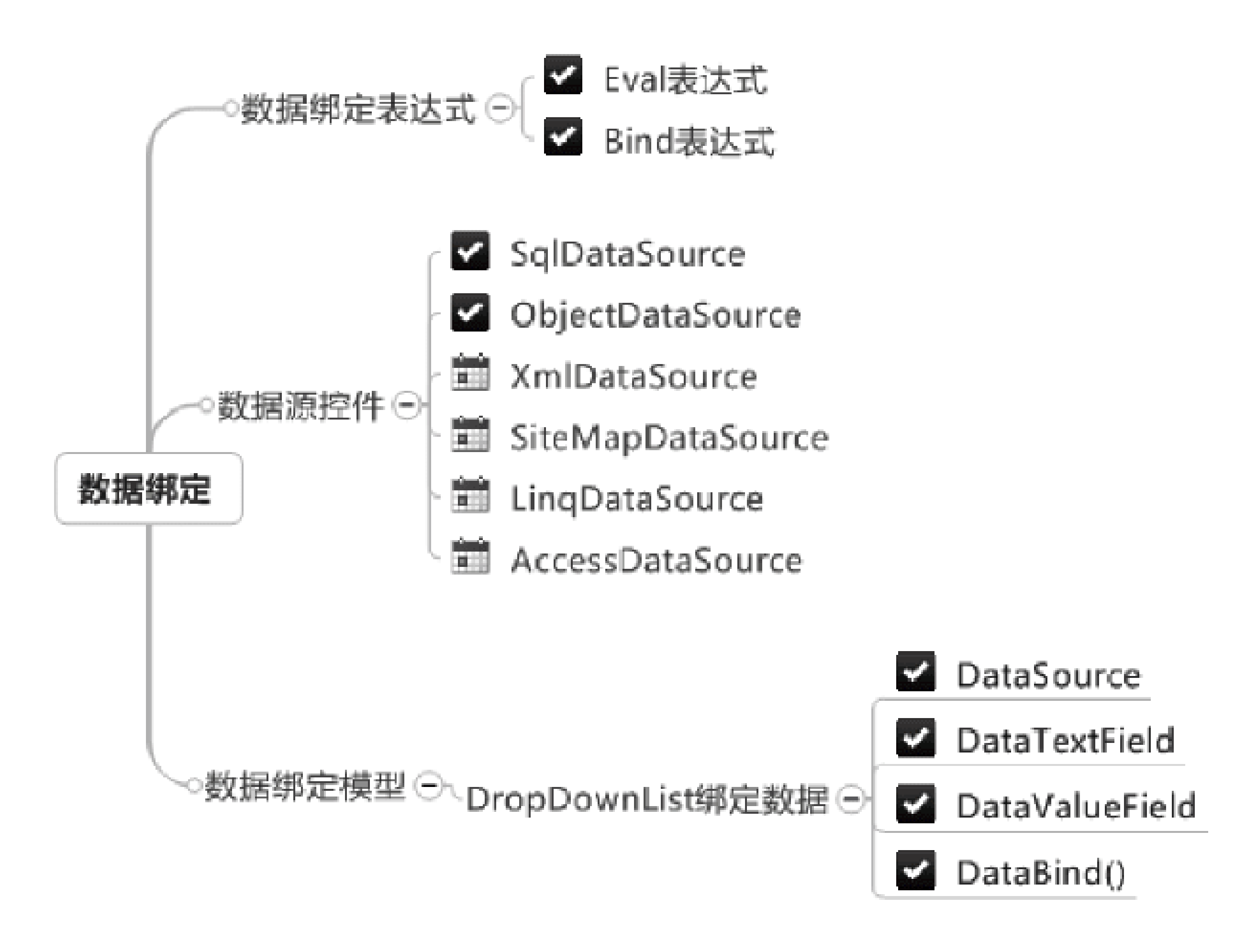


图 7-11　数据绑定知识体系

7.5　单元练习题

一、选择题

1. 关于 SqlDataSource 数据源控件相关属性，说法不正确的是(　　)。

A. 该控件的 ProviderName 属性表示 SqlDataSource 控件连接数据库的提供程序名称

B. ConnectionString 属性表示 SqlDataSource 控件可使用该参数连接到数据库，但是不能从应用程序的配置文件中读取

C. SelectCommand 属性表示 SqlDataSource 控件从数据库中选择数据所使用的 SQL 命令

D. ControlParameter 实际是个控件，在代码中应改写成<asp：ControlParameter>，使用特定控件的值

2. 使用 3 层架构实现表示层显示学员信息，学员信息中包含的年级(Grade)对象作为一个属性。现在将要显示学员的年级名称(gradeName)，下列绑定语句正确的是(　　)。

A. <%#Bind("GradeName")%>　　B. <%#Bind("Grade.gradeName")%>

C. <%#Eval("GradeName")%>　　D. <%#Eval("Grade.gradeName")%>

二、填空题

1. 数据绑定表达式包含在<%#　%>分隔符之内，并使 Eval 和 Bind 方法。________方法用于定义单向(只读)绑定。________方法用于定义双向(可更新)绑定。

2. ObjectDataSource 控件使开发人员能够在保留 3 层应用程序结构的同时，使用 ASP.NET 数据源控件。完成下面为 ObjectDataSource 控件定义好的 Insert 方法。

```
public void Insert(int id, string name){
        string strcnn = ConfigurationManager.ConnectionStrings
        ["StudentCnnString"].ConnectionString;
        using (SqlConnection sqlConn = new SqlConnection(strcnn)){
        string insertString = "insert into Major values(" + id + ",'" + name + "')";
```

```
        SqlCommand sqlCmd = sqlConn. CreateCommand;     //创建 SqlCommand 对象
        sqlCmd.CommandText = ________;
        sqlConn.Open();
        sqlCmd.ExecuteNonQuery();
        sqlConn.Close();
    }
}
```

三、问答题

1. 试说明什么是数据源控件。ASP.NET 3.5 中提供了几种数据源控件?
2. 比较 SqlDataSource、ObjectDataSource 和 SiteMapDataSource 控件的使用。

单元 8 数据绑定控件的应用

教学目标：

- 熟练使用数据源控件和数据绑定控件在 Web 页面中输出数据，对后台数据库中的数据进行修改和更新。
- 掌握在 Web 页面中灵活使用 ADO.NET 对象、数据源控件和数据绑定控件实现应用程序功能。
- 掌握 SqlDataSource、ObjectDataSource 数据源控件和 GridView、DataList、Repeater、DetailsView、DataPager 等数据绑定控件的功能、属性和事件。

8.1 知识准备

8.1.1 数据绑定控件

1. 数据绑定控件的层次结构

数据源控件并不能显示数据，将数据显示出来需要数据绑定控件。数据绑定控件的层次结构如图 8-1 所示，从中可以看出数据绑定控件跟数据源控件一样可以分为两大类：普通绑定控件(DataBoundControl)和层次化绑定控件(HierachicalDataBoundControl)。其中

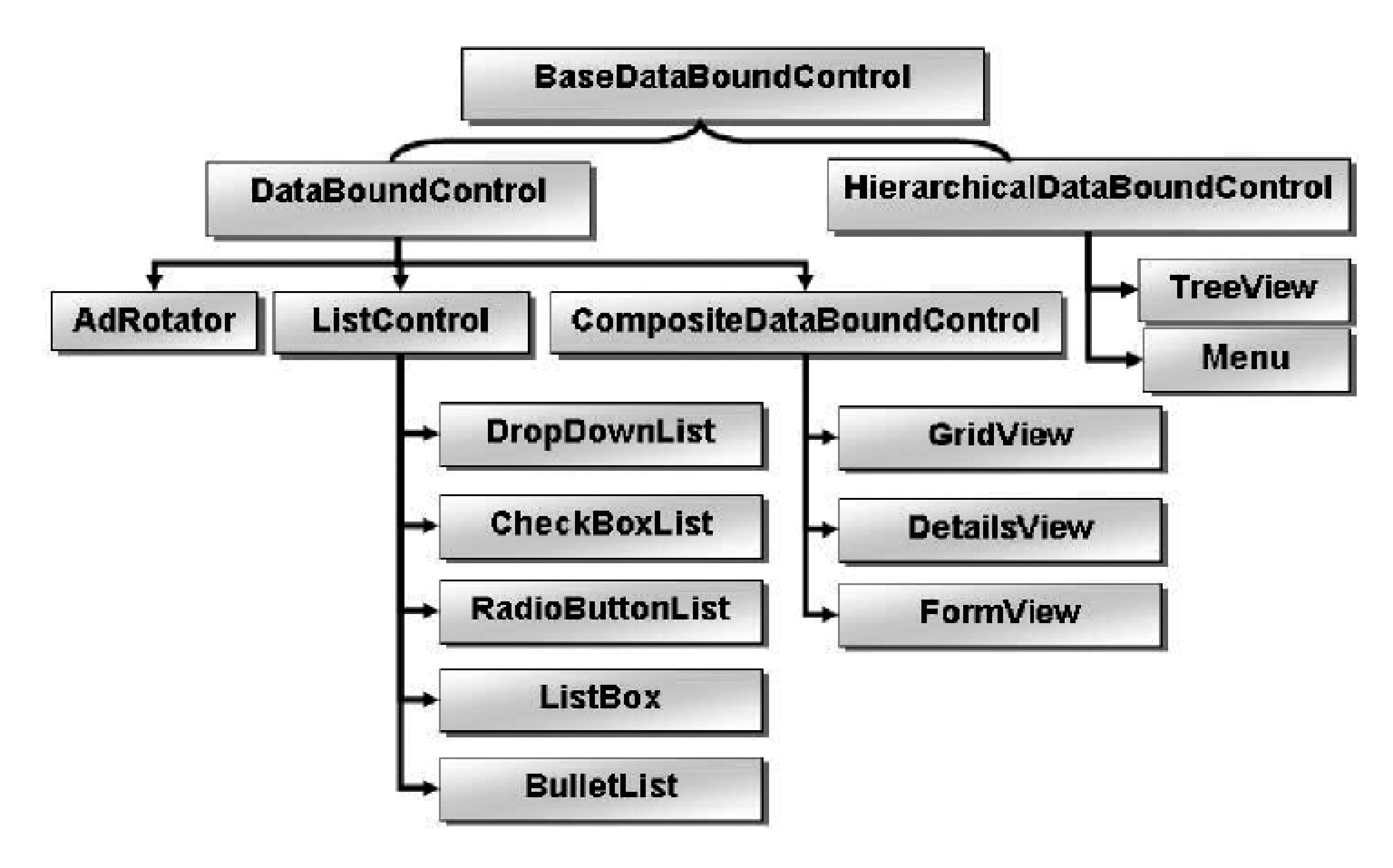


图 8-1　数据绑定控件的层次结构

普通绑定控件又分为标准型控件、列表控件和复合型控件(CompositeDataBoundControl)。通常复合型控件用于表格显示。

表8-1对部分常见的数据绑定控件进行了简单介绍。

表8-1　常见的数据绑定控件

控件名称	说　明
DropDownList	下拉列表控件,比如实现图书分类的修改,分类可以使用下拉菜单的形式给用户提供选择
GridView	通过表格方式实现数据的展示,其中每列表示一个字段,每行表示一条记录。比如显示图书列表
DetailsView	显示单条记录的详细信息,并支持对记录的添加、删除、修改等。比如显示图书的详细页,可以使用该控件

2. 数据绑定控件与数据源控件

通俗地讲,数据绑定就是把数据源中的数据取出来,显示在窗体的各种控件上,用户可以通过这些控件查看和修改数据,这些修改会自动地保存到数据源中。要使数据绑定控件显示有用的内容,则需要为它们指定数据源(Data Source)。要将这一数据源绑定到控件,可以使用一个单独的数据源控件来为数据绑定控件管理数据。

要执行绑定,应将数据绑定控件的DataSourceID属性设置为SqlDataSource、ObjectDataSource、SiteMapDataSource等数据源控件。数据源控件连接到数据库、实体类或中间层对象等数据源,然后检索或更新数据,最后,数据绑定控件即可使用这些数据。当数据绑定控件绑定到数据源控件时,无须编写代码或者只需要编写少量额外代码即可执行数据操作。数据绑定控件可以自动利用数据源控件提供的数据服务。

除了设置数据绑定控件的DataSourceID属性指定数据源外,还可以通过编写代码在程序运行中动态绑定数据源。

1. 指定数据源控件方式

语法格式为:

```
数据绑定控件 ID.DataSourceID = 数据源控件 ID;
```

例如:

```
<asp:GridView ID="GridView1" runat="server"  AutoGenerateColumns="False"
          DataSourceID="SqlDataSource1"  EmptyDataText="没有可显示的记录">
```

或

```
if (!IsPostBack)
{
   GridView1.DataSourceID = "SqlDataSource1"; //SqlDataSource1 为数据源控件对象。
}
```

2. 编码指定数据源方式

语法格式为：

```
数据绑定控件 ID.DataSource = 数据集合;
数据绑定控件 ID.DataBind();
```

例如：

```
this.gvMain.DataSource = new UserManager().GetNormalUsers();
this.gvMain.DataBind();
```

其中，gvMain 是数据绑定控件 GridView，业务逻辑层的 GetNormalUsers()方法返回的是类型为 List<User>的所有正常用户数据。

释疑：在 ASP.NET 中，指定数据源控件的方式和编码指定数据源的方式的区别。

(1) 语法不同，指定数据源控件的方式只要指定数据源控件 ID 后即可完成绑定，但编码指定数据源的方式除了指定 DataSource 属性外还需要使用绑定方法 DataBand()才能完成数据绑定，DataBand()方法是将数据源绑定到被调用的服务器控件及其所有子控件上。

(2) 指定数据源控件的方式可以使用数据源控件的功能，比如更新、删除等(需要指定相关方法)，但编码指定数据源的方式只能提供绑定的显示。

ASP.NET 包含了很多支持简单数据绑定的控件，如 TextBox、Label、ListControl、CheckBoxList、RadioButtonList、DropDownList 等控件，单元 7 中曾详细讲解过 RadioButtonList、DropDownList 的使用，简单数据绑定控件通常只显示单个值。ASP.NET 中的复杂数据绑定控件包括 GridView、DataList、Repeater、DetailsView、FormView、ListView 等，复杂数据绑定控件与简单数据绑定控件的区别在于它们可以用更精细的方式来显示数据。下面介绍 GridView、DataList、Repeater、DetailsView、DataPager 等复杂数据控件。

8.1.2 GridView 控件

GridView 控件通常与数据源控件结合使用，以表格的形式显示数据库中的数据，可以对记录中的行实现删除、修改、选择和分页功能，可以对列实现排序功能。在默认情况下，GridView 通过 SqlDataSource 访问数据库，可以访问多种关系数据库，也可以读取 XML 文件。GridView 控件的常用属性如表 8-2 所示。

表 8-2 GridView 控件的常用属性

属 性	说 明
AllowPaging	指示是否启用分页功能
AllowSorting	指示是否启用排序功能
AutoGenerateColumns	指示是否为数据源中的每个字段自动创建绑定字段
AutoGenerateDeleteButton	指示每个数据行是否添加“删除”按钮
AutoGenerateEditButton	指示每个数据行是否添加“编辑”按钮

续表

属　　性	说　　明
AutoGenerateSelectButton	指示每个数据行是否添加“选择”按钮
EditIndex	获取或设置要编辑行的索引
DataKeyNames	获取或设置 GridView 控件中的主键字段的名称。多个主键字段间，以逗号隔开
DataKeys	获取 GridView 中使用 DataKeyNames 设置的每一行主键值的对象集合
DataSource	获取或设置对象，数据绑定控件从该对象中检索其数据项列表
DataMember	当数据源有多个数据项列表时，获取或设置数据绑定控件绑定到的数据列表的名称
DataSourceID	获取或设置控件的 ID，数据绑定控件从该控件中检索其数据项列表
PageCount	获取在 GridView 控件中显示数据源记录所需的页数
PageIndex	获取或设置当前显示页的索引
PageSize	获取或设置每页显示的记录数

下面通过两种方式了解 GridView 的数据绑定。

1. 使用数据源控件

使用数据源控件即通过设置 GridView 控件的 DataSourceID 属性将数据源控件绑定到控件上。下面的示例就是使用 GridView 做一个显示图书信息列表的页面，除了显示，还有排序、分页等功能。

【示例 8-1】 使用 GridView 数据绑定控件(结合 SqlDataSource 数据源)实现图书信息列表的显示、排序及分页功能。

1) 配置 SqlDataSource 数据源

(1) 将“新知书店”数据库文件 BookShopPlus. mdf 附加到 SQL Server 2005 数据库中。

(2) 新建一个名为 ch8_1 的网站，右击网站添加页面 BookList. aspx。

(3) 切换到页面 BookList. aspx 的“设计”视图，拖放一个 SqlDataSource 控件到页面中，其默认 ID 为 SqlDataSource1，右击 SqlDataSource 控件，在弹出的快捷菜单中，选择“属性”命令，修改其 ID 属性值为 sdBook。

(4) 选择 SqlDataSource 控件，单击右上方的箭头符号，选择“配置数据源”选项，弹出“配置数据源”对话框，如图 8-2 所示。

(5) 单击“新建连接”按钮，弹出“新建连接”对话框，如图 8-3 所示，在“服务器名”的下拉列表框中输入 localhost、. 或本地计算机名字。在“选择或输入一个数据库名”的下拉列表框中选中 BookShopPlus 数据库，可单击“添加连接”对话框中的“测试连接”按钮，看数据库是否成功连接。若不成功，则说明服务器或者数据库名选择或者输入错误。

(6) 如图 8-3 所示，单击两次“确定”按钮，此时“新建连接”成功的“配置数据源”对话框如图 8-4 所示。

(7) 单击“下一步”按钮，如图 8-5 所示，在“配置数据源”向导的“将连接字符串保存到应用程序配置文件中”界面选中“是，将此连接另存为”复选框。确认将连接字符串保存到配置文件中，另存为 BookDbConnString，在下次连接时可以直接使用。另外，将连接字符串和查询字符串写入 web. config 配置文件也能简化工作，程序代码也更清晰。



图 8-2 “配置数据源”对话框



图 8-3 “添加连接”对话框



图 8-4　配置数据源窗口连接成功后



图 8-5　“将连接字符串保存到应用程序配置文件中”界面

（8）单击“下一步”按钮，在“配置 Select 语句”界面中指定需要检索的数据表及其字段，这里选择 Books 表，选中“ * ”复选框及所有字段，如图 8-6 所示。



图 8-6 “配置 Select 语句”界面

（9）单击“高级”按钮，弹出“高级 SQL 生成选项”对话框，选中“生成 INSERT、UPDATE 和 DELETE 语句”复选框，如图 8-7 所示。单击“确定”按钮返回“配置 Select 语句”界面，单击 ORDER BY 按钮，在打开的“添加 ORDER BY 子句”对话框中将“排序方式”设置为 UnitPrice 字段降序排列，如图 8-8 所示。单击“确定”按钮返回“配置 Select 语句”界面。



图 8-7 “高级 SQL 生成选项”对话框

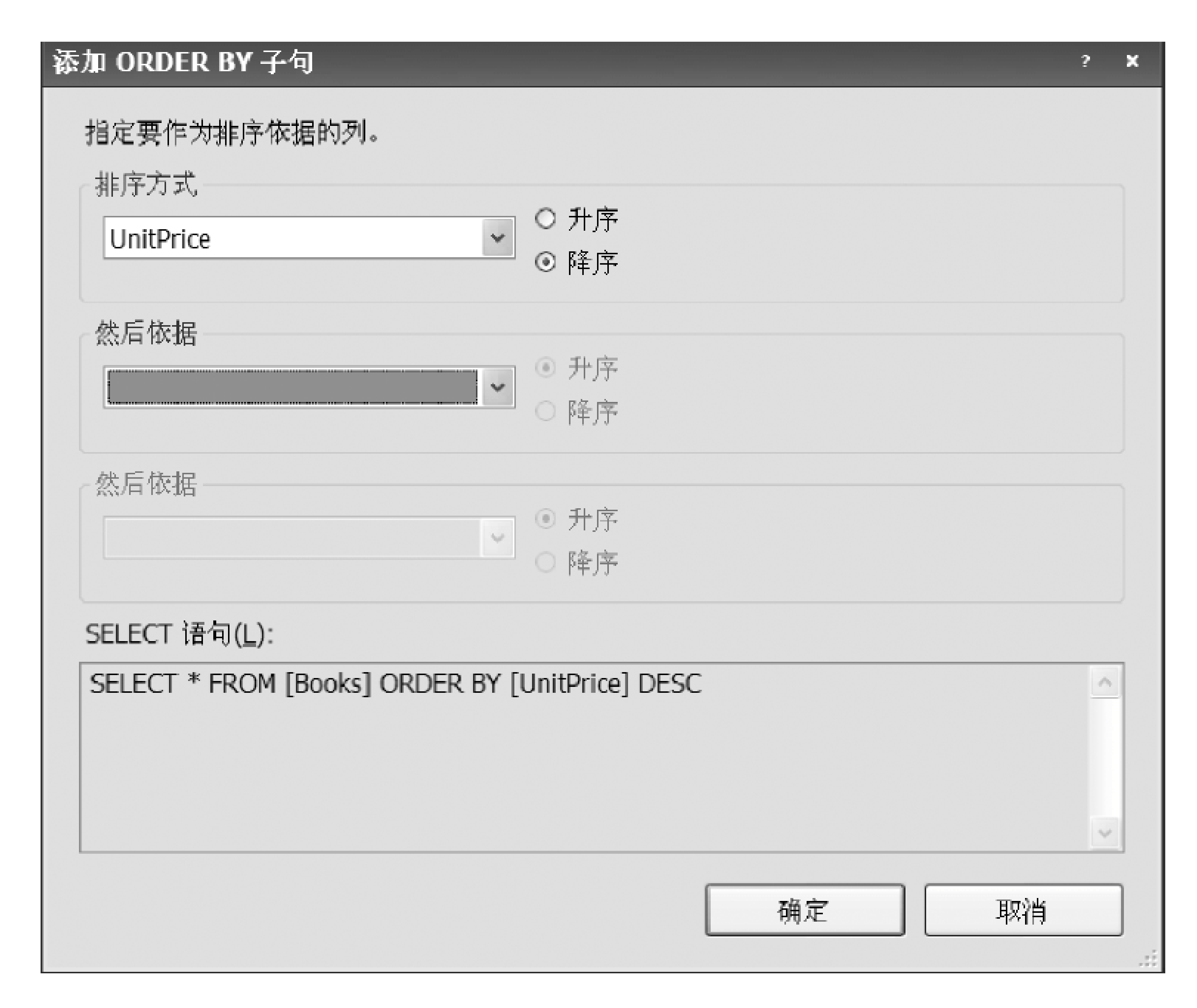


图 8-8　“添加 ORDER BY 子句”对话框

(10) 单击“下一步”按钮，在“测试查询”窗口中可以看到配置的 Select 语句效果，如图 8-9 所示，单击“完成”按钮完成对数据源的配置。



图 8-9　“测试查询”窗口

通过上述步骤，实现了将一个 SqlDataSource 控件与 SQL Server 数据源的连接。在整个过程中无须编写代码，降低了 Web 数据库编程的难度。表 8-3 给出本例中连接到一个 SQL Server 数据库的 SqlDataSource 控件的示例代码，这些代码是完成上述步骤后自动在 BookList.aspx 页面中生成的。

表 8-3 已配置的 SqlDataSource 数据源对应的代码

行号　　代　码　页

```
<asp:SqlDataSource ID = "sdBook" runat = "server"
    ConnectionString = "<%$ ConnectionStrings:BookDbConnString %>"
    DeleteCommand = "DELETE FROM [Books] WHERE [Id] = @Id"
    InsertCommand = "INSERT INTO [Books] ([Title], [Author], [PublisherId], [PublishDate], [ISBN], [UnitPrice],
                [ContentDescription], [TOC], [CategoryId], [Clicks]) VALUES (@Title, @Author,
                @PublisherId, @PublishDate, @ISBN, @UnitPrice, @ContentDescription, @TOC,
                @CategoryId, @Clicks)"
    SelectCommand = "SELECT * FROM [Books] ORDER BY [UnitPrice] DESC"
    UpdateCommand = "UPDATE [Books] SET [Title] = @Title, [Author] = @Author,
                [PublisherId] = @PublisherId, [PublishDate] = @PublishDate, [ISBN] = @ISBN,
                [UnitPrice] = @UnitPrice, [ContentDescription] = @ContentDescription,
                [TOC] = @TOC, [CategoryId] = @CategoryId, [Clicks] = @Clicks WHERE [Id] = @Id">
    <DeleteParameters>
        <asp:Parameter Name = "Id" Type = "Int32" />
    </DeleteParameters>
    <InsertParameters>
        <asp:Parameter Name = "Title" Type = "String" />
        <asp:Parameter Name = "Author" Type = "String" />
        <asp:Parameter Name = "PublisherId" Type = "Int32" />
        <asp:Parameter Name = "PublishDate" Type = "DateTime" />
        <asp:Parameter Name = "ISBN" Type = "String" />
        <asp:Parameter Name = "UnitPrice" Type = "Decimal" />
        <asp:Parameter Name = "ContentDescription" Type = "String" />
        <asp:Parameter Name = "TOC" Type = "String" />
        <asp:Parameter Name = "CategoryId" Type = "Int32" />
        <asp:Parameter Name = "Clicks" Type = "Int32" />
    </InsertParameters>
    <UpdateParameters>
        <asp:Parameter Name = "Title" Type = "String" />
        <asp:Parameter Name = "Author" Type = "String" />
        <asp:Parameter Name = "PublisherId" Type = "Int32" />
        <asp:Parameter Name = "PublishDate" Type = "DateTime" />
        <asp:Parameter Name = "ISBN" Type = "String" />
        <asp:Parameter Name = "UnitPrice" Type = "Decimal" />
        <asp:Parameter Name = "ContentDescription" Type = "String" />
```

续表

行号	代　码　页
36	< asp:Parameter Name = "TOC" Type = "String" />
37	< asp:Parameter Name = "CategoryId" Type = "Int32" />
38	< asp:Parameter Name = "Clicks" Type = "Int32" />
39	< asp:Parameter Name = "Id" Type = "Int32" />
40	</UpdateParameters >
41	</asp:SqlDataSource >

2）显示内容的选择

切换到页面 BookList.aspx 的“设计”视图，拖放一个 GridView 控件到页面中，修改其 ID 属性值为 gvBook，在“GridView 任务”窗口的“选择数据源“下拉列表框中选择 sdBook 选项。发现所有字段都显示出来了，如图 8-10 所示，但用户只想显示书名、作者和价格。

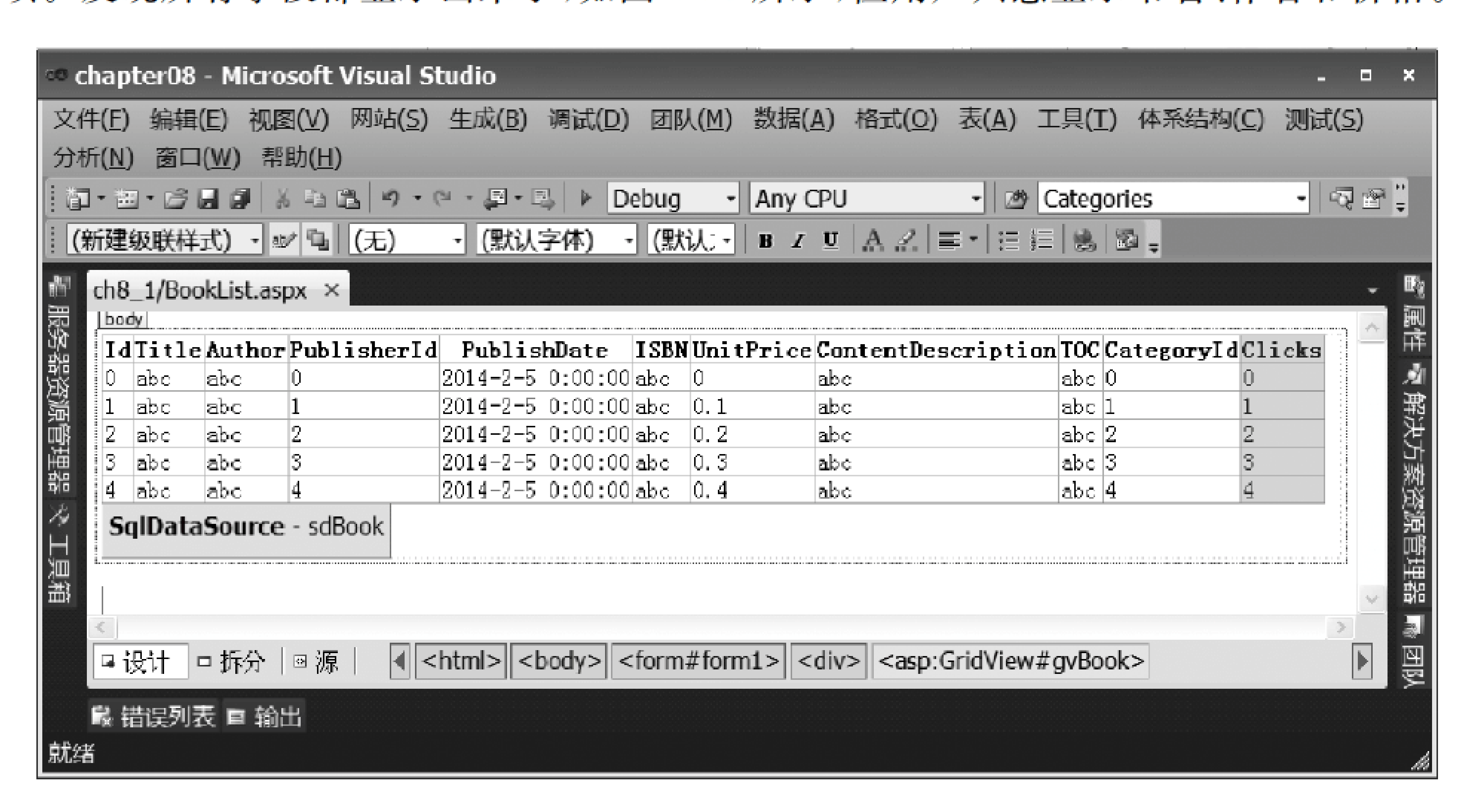


图 8-10　GridView 控件绑定数据源窗后的效果

选中 GridView 控件，单击右上方的箭头，打开“GridView 任务”窗口，如图 8-11 所示，在该窗口中选择“编辑列”选项，打开“字段”对话框，如图 8-12 所示。

GridView 任务
自动套用格式...
选择数据源: sdBook
配置数据源...
刷新架构
编辑列...
添加新列...
左移列
移除列
启用分页
启用排序
启用编辑
启用删除
启用选定内容
编辑模板

图 8-11　编辑列

在“字段”对话框可以设置显示的内容、标题以及顺序，如图 8-12 所示。在下侧的上下箭头按钮可以调整列的显示顺序，下箭头下面的按钮作用是删除。当选中某个字段时，在右侧的字段属性中可设置相应属性。此时只需要修改显示的标题头即可，即在 Header Text 栏的属性值中填写对应的中文标题。

3）设置分页和排序

显示书籍信息列表一般内容比较多，需要设置分页，如图 8-13 所示，选中“启用分页”复选框即可。还可以通过属性窗体中的 PageSize 属性设置每页显示多少条记录，如图 8-14 所示，AllowPaging 属性是设置是否打开分页功能，PageIndex 设



图 8-12 “字段”对话框

置的是当前显示第几页，0 页是第一页，PageSize 属性是设置每页显示多少条，默认是 10 条，这里改成 4 条。

GridView 任务
自动套用格式...
选择数据源: sdBook
配置数据源...
刷新架构
编辑列...
添加新列...
左移列
移除列
☑ 启用分页
☑ 启用排序
☐ 启用编辑
☐ 启用删除
☐ 启用选定内容
编辑模板

图 8-13 设置分页

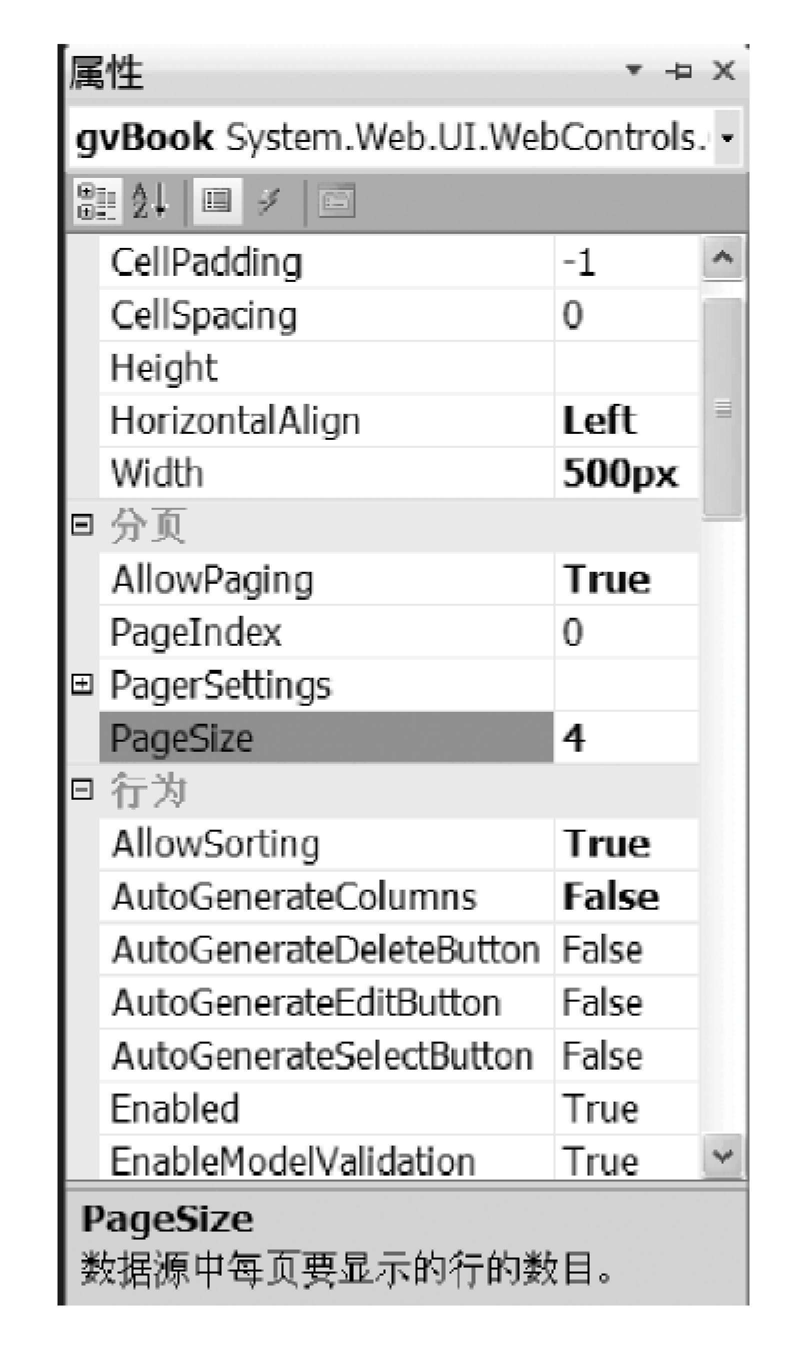


图 8-14 分页和排序

启动排序就是按图 8-13 选中“启用排序”复选框。启用排序就是在单击标题头时，比如单击“单价”时，就会按照单价所对应的字段进行排序

最后，还可以将显示的数据应用格式，Visual Studio 提供了如图 8-15 所示的几种样式，可以方便地套用，在图 8-13 中选择“自动套用格式”选项，即可弹出“自动套用格式”对话框，如图 8-15 所示。

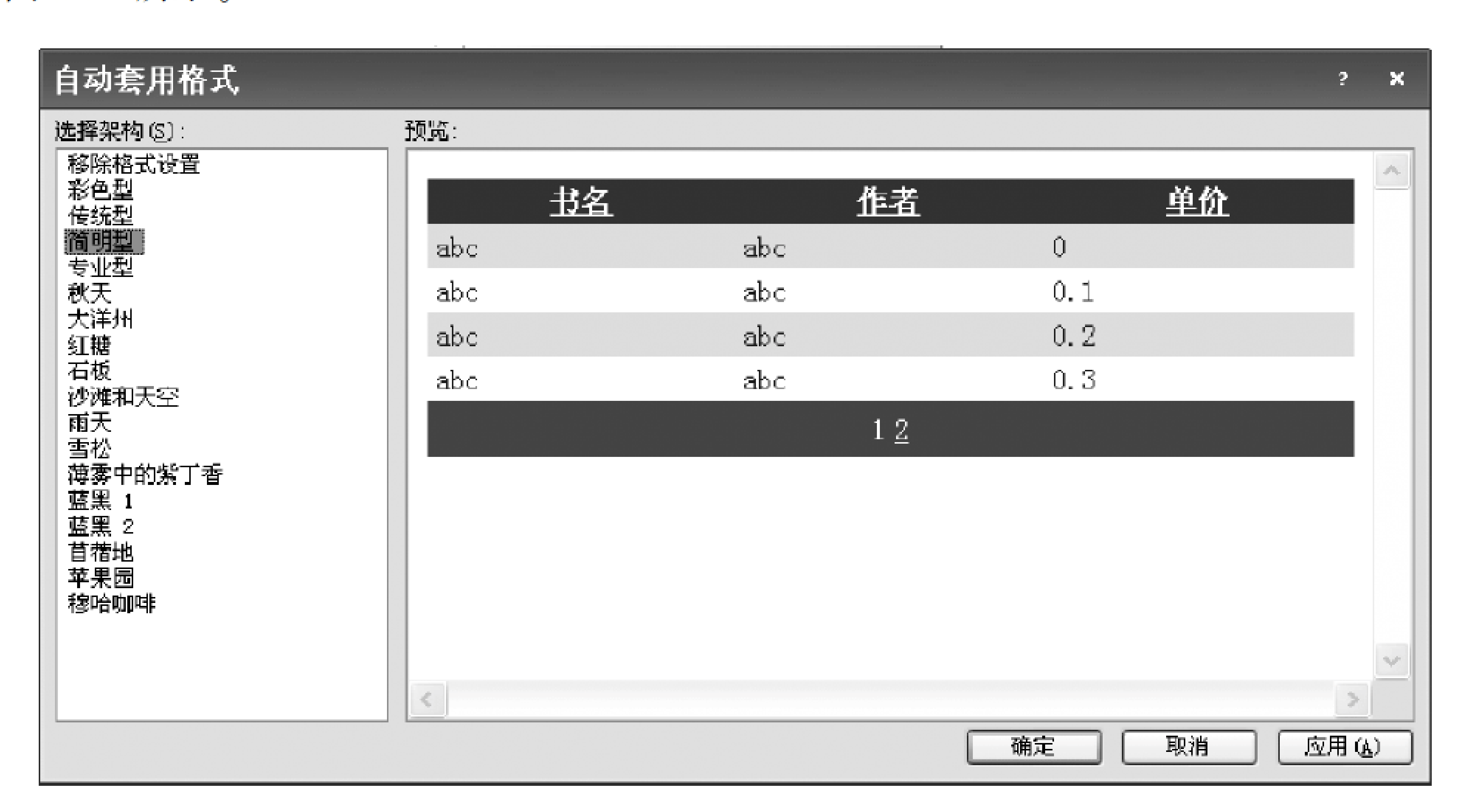


图 8-15 “自动套用格式”对话框

运行 BookList.aspx 页面，效果如图 8-16 所示。

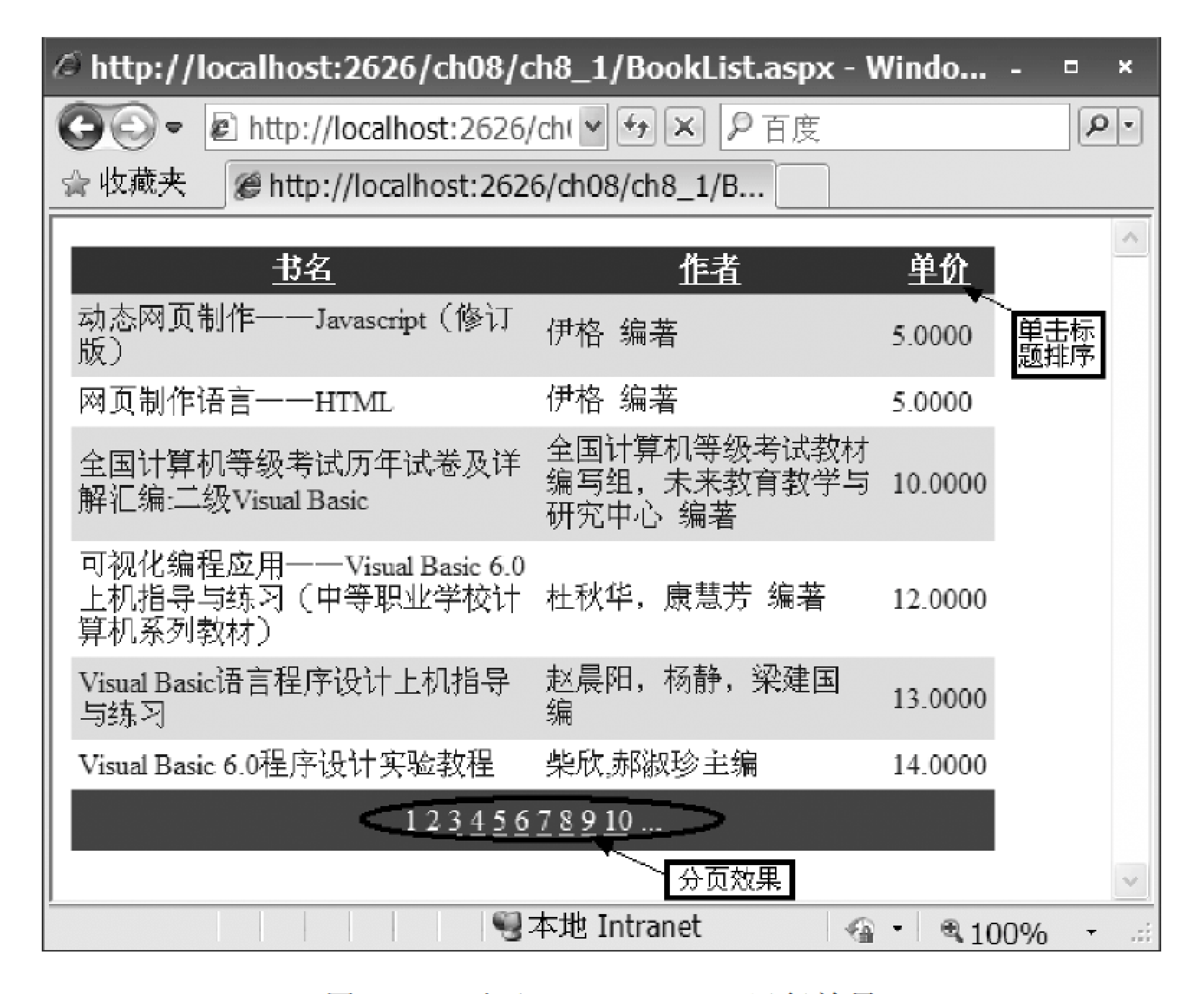


图 8-16 页面 BookList.aspx 运行效果

整个步骤没有编写一行代码，不仅实现了显示、排序、分页的功能，还有不错的显示效果。查看视图中的代码，可以发现，SqlDataSource 控件通过 GridView 控件的 DataSourceID 属性绑定到 GridView。

2. 编码指定数据源

上面使用数据源控件方式轻松实现了图书信息的显示、排序和分页功能，但此种方式灵活性不够，在实际项目中很少使用。下面以“新知书店”图书列表的显示为例来介绍编码指定数据源方式，此方式要通过手写代码实现排序、分页、更新和删除等操作，由于是手工编写，因此灵活性高，实际项目中也基本采用这种方式。

【示例 8-2】 用编码指定数据源方式实现新知书店(管理端)图书列表的显示。

1) 图书列表显示数据访问层与业务逻辑层的实现

图书列表显示数据访问层与业务逻辑层的工作主要就是对图书信息表(Books)进行查询并返回一个图书列表对象集合。

(1) 图书列表数据访问层的实现。

在 BookShopDAL 项目的 BookService.类中，添加一个获取图书列表对象的方法，代码如表 8-4 所示。

表 8-4 图书列表显示数据访问层 BookService 类代码

```
public class BookService
{
    string connection = ConfigurationManager.ConnectionStrings["BookShop"].ConnectionString;
    /// <summary>
    /// 获取所有图书信息列表
    /// </summary>
    /// <returns>图书列表对象</returns>
    public List<Book> GetBooks()
    {
        string sqlAll = "SELECT * FROM Books";
        return GetBooksBySql(sqlAll);
    }
    /// <summary>
    /// 根据查询字符串获取图书列表对象
    /// </summary>
    /// <param name="safeSql">SQL 语句</param>
    /// <returns>图书列表对象</returns>
    public List<Book> GetBooksBySql(string safeSql)
    {
        List<Book> list = new List<Book>();
        DataSet ds = SqlHelper.ExecuteDataset(this.connection, CommandType.Text, safeSql);
        if (ds.Tables.Count > 0)
        {
```

续表

行号　代码页

```
            DataTable dt = ds.Tables[0];
            foreach (DataRow row in dt.Rows)
            {
                Book book = new Book();
                book.Id = (int)row["Id"];
                book.Title = (string)row["Title"];
                book.Author = (string)row["Author"];
                book.PublishDate = (DateTime)row["PublishDate"];
                book.ISBN = (string)row["ISBN"];
                book.UnitPrice = (decimal)row["UnitPrice"];
                book.ContentDescription = (string)row["ContentDescription"];
                book.TOC = (string)row["TOC"];
                book.Clicks = (int)row["Clicks"];
                book.Publisher = new PublisherService().GetPublisherById((int)row["PublisherId"]); //FK
                book.Category = new CategoryService().GetCategoryById((int)row["CategoryId"]); //FK
                list.Add(book);
            }
        }
        return list;
    }
}
```

（2）图书列表业务逻辑层的实现。

在 BookShopBLL 项目的 BookManager 类中，添加相应方法，代码如下。

```
public class BookManager
{
    public List<Book> GetBooks()
    {
        return new BookService().GetBooks();
    }
}
```

2）图书列表的绑定

（1）将 GridView 控件拖入解决方案 BookShop 下 Web 网站项目中的 admin 文件夹的 BookList.aspx 页面中，将其 ID 属性值修改为 gvBooks，如图 8-17 所示。

（2）切换到 BookList.aspx.cs 文件，在 BookList.aspx 页面的 Page_Load 事件方法中编写如下代码，实现数据和 GridView 控件的绑定。

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
        if (!IsPostBack)
        {
            this.gvBooks.DataSource = new BookManager().GetBooks();
            this.gvBooks.DataBind();
        }
    }
```

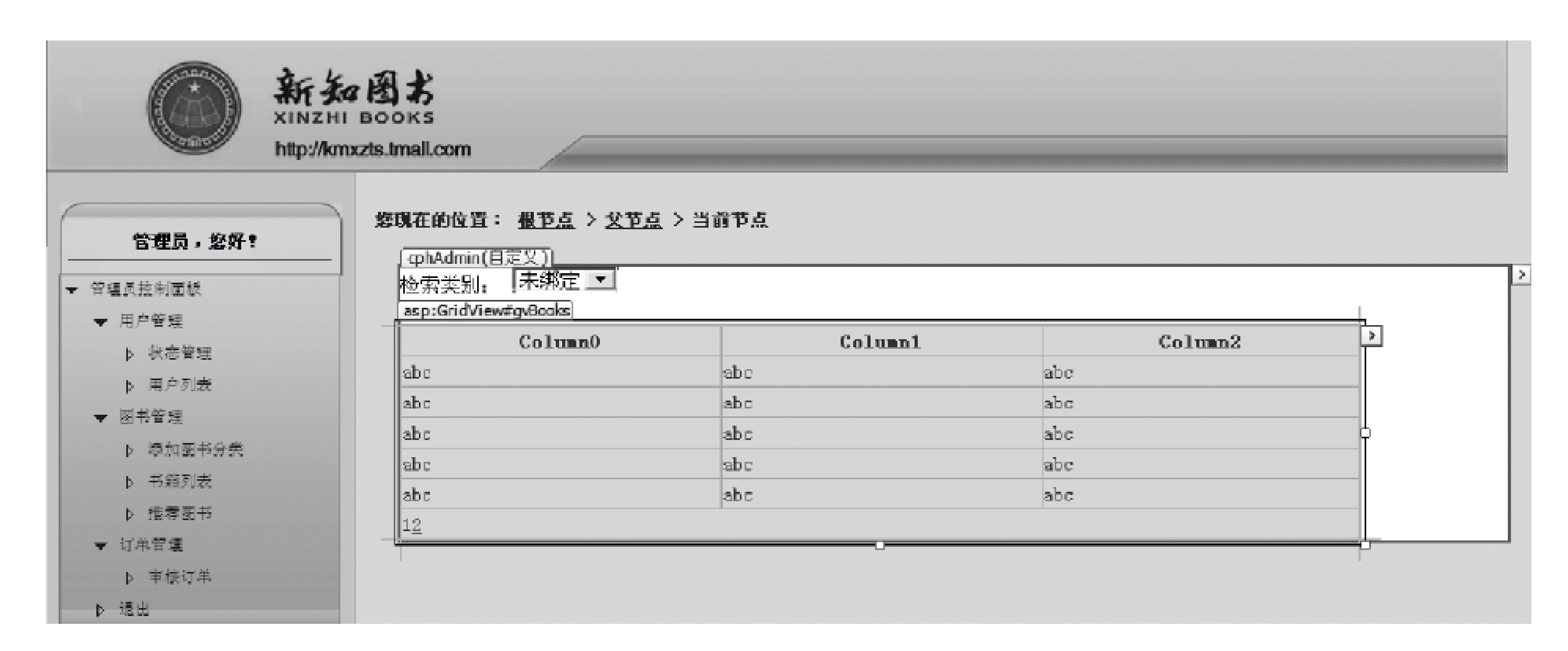


图 8-17　将 GridView 控件拖入 BookList.aspx 内容页的效果

同 DropDownList 的数据绑定一样，这里也要先指定数据源（即指定 DataSource）为 GetBooks()方法返回的 List<Book>集合，再使用 DataBind()方法实现绑定。

(3) 运行代码后发现数据全部显示出来，且 Header 显示的是数据库中的字段名称，格式也不美观，现在来完善数据显示，先选择需要显示的列，如图 8-18 所示。



图 8-18　选择绑定列

添加 BoundField 字段，在 DataField 属性中设置显示的字段。注意，如果仅仅设置显示的字段，会发现系统又重复显示了一遍所有的数据，解决的办法是将 GridView 的 AutoGenerateColumns 属性设置为 False 或取消选中图 8-18 中的“自动生成字段”复选框。AutoGenerateColumns 属性表示是否为数据源中的每一字段自动创建 BoundColumn 对象并在 GridView 控件中显示这些对象。

（4）设置分页，首先将 GridView 控件的 AllowPaging 属性设置为 True，设置成允许分页，然后编写 GridView 控件的 PageIndexChanging 事件方法代码如下：

```
/// <summary>
/// gvBooks 对象的 PageIndexChanging 事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void gvBooks_PageIndexChanging(object sender, GridViewPageEventArgs e)
{
    this.gvBooks.PageIndex = e.NewPageIndex;
    //需要重新绑定图书列表
    this.gvBooks.DataSource = new BookManager().GetBooks();
    this.gvBooks.DataBind();
}
```

其中，NewPageIndex 属性表示页面选中的页索引。

3）绑定字段的设置

刚才在选择绑定列时，选择的都是 BoundField 字段，其实 GridView 控件提供了 7 种数据绑定列的类型。

（1）BoundField：用于显示普通文本，是默认的数据绑定列的类型，一般自动生成的列就是该类型。BoundField 字段的常用属性如表 8-5 所示，需要注意的是，DataFormatString 属性可以设置显示的格式，常见的格式有以下几种：

- {0:C}，设置显示的内容是货币类型。
- {0:N}，设置显示的内容是数字。
- {0:yy-MM-dd}，设置显示的是日期格式。

表 8-5　BoundField 字段的常用属性

属　性	说　明
DataField	指定列将要绑定字段的名称，如果是数据表则为数据表的字段；如果是对象，则为该对象的属性
DataFormatString	用于格式化 DataField 显示的格式化字符串。例如如果需要指定 4 位小数，则格式化字符串为{0: F4}；如果需要指定为日期，则格式字符串为{0:d}
ApplyFormatInEditMode	是否将 DataFormatString 设置的格式应用到编辑模式
HeaderText、FooterText、HeaderImageUrl	前两个用于设置列头和列尾区显示的文本。HeaderText 属性通常用于显示列名称。列尾可以显示一些统计信息
ReadOnly	列是否只读，默认情况下，主键字段是只读，只读字段将不能进入编辑模式

续表

属　性	说　明
Visible	列是否可见。如果设置为 false,则不产生任何 HTML 输出
HtmlEncode	HtmlEncode 指定是否对显示的文本内容进行 HTML 编码,默认值为 true

(2) TemplateField:它允许以模板形式自定义数据绑定列的内容。它是这几种绑定列中最灵活的表现形式,但也是最复杂的,甚至可能需要编写 HTML 代码,GridView 控件的模板列如表 8-6 所示。

表 8-6　GridView 控件的模板列

模　板	说　明
AlternatingItemTemplate	为交替项指定要显示的内容
EditItemTemplate	为处于编辑模式中的项指定要显示的内容
FooterTemplate	为对象的脚注部分指定要显示的内容
HeaderTemplate	为标头部分指定要显示的内容
InsertItemTemplate	为处于插入模式中的项指定要显示的内容。只有 DetailsView 控件支持该模板
ItemTemplate	为 TemplateField 对象中的项指定要显示的内容

从表 8-6 可看出,GridView 控件的模板列有普通项(ItemTemplate)、交替项(AlternatingItemTemplate)、编辑项(EditItemTemplate)、插入项(InsertItemTemplate)、脚注(FooterTemplate)、标头(HeaderTemplate),GridView 本身还提供空数据项(EmptyDataTemplate)、页导航(PageTemplate),可以根据需要选择要设置的模板。

模板字段的添加有两种方式:直接添加或者将现有字段转换为模板字段。首先使用直接添加方式加入多选的列,如图 8-19 所示。

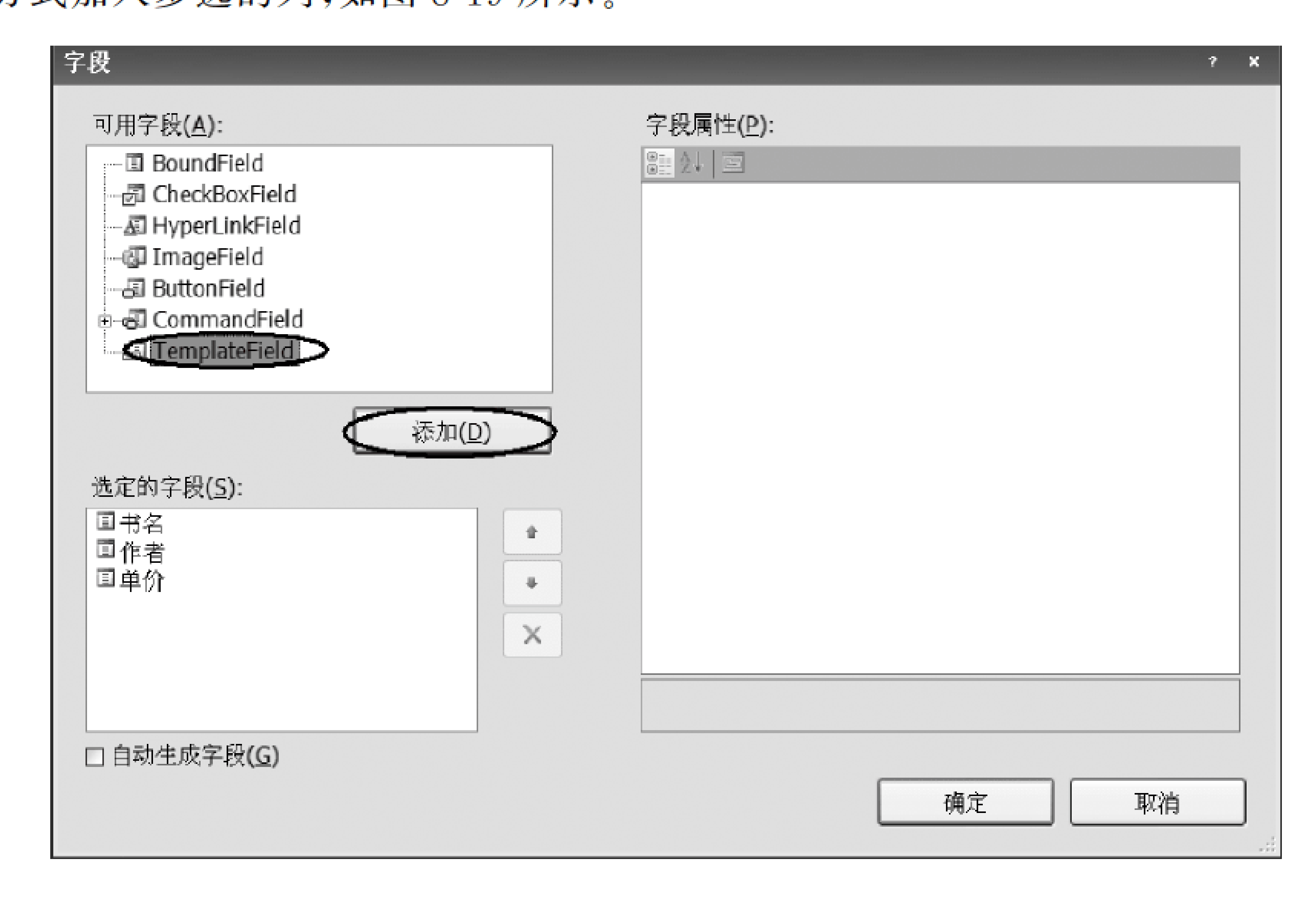


图 8-19　添加模板列

添加模板列后，可以通过单击 GridView 控件的选项按钮，编辑模板，如图 8-20 所示。



图 8-20　编辑模板列

可以看到，模板列也可以将控件拖入页面，这样编辑起来就比较方便。如果不喜欢这种方式，可以直接在模板中编写 HTML 如下代码：

```
<asp:TemplateField HeaderText = "全选">
    <HeaderTemplate>
        <input type = "checkbox" />全选
    </HeaderTemplate>
    <ItemTemplate>
        <asp:CheckBox ID = "chbSelect" runat = "server" />
    </ItemTemplate>
    <ControlStyle Width = "50px" />
</asp:TemplateField>
```

这段代码就是在图 8-21 中看到的模板列的代码，这里只设置了普通项模板和标题头模板。



图 8-21　编辑模板列界面

刚才只是在模板列中显示了一个固定的内容，但用户经常需要在列中展现数据。下面介绍自动转换的模板列，将自动生成的书名列转换为模板列，如图 8-22 所示。

转换后的代码如下：

```
<asp:TemplateField HeaderText = "书名">
    <EditItemTemplate>
```

```
            //注意单引号的使用
            <asp:TextBox ID="TextBox1" runat="server" Text='<%# Bind("Title") %>'></asp:
    TextBox>
        </EditItemTemplate>
        <ItemTemplate>
            //绑定字段的方法,可以调用 Eval()或 Bind()
            <asp:Label ID="Label1" runat="server" Text='<%# Bind("Title") %>'></asp:Label>
        </ItemTemplate>
    </asp:TemplateField>
```



图 8-22　转换成模板列

使用<%# Bind("字段名") %>就可以实现在模板列中展示数据。若要灵活显示内容的列,可以使用模板列。那么如果绑定的是对象怎么办呢?比如图书的分类,在图书的属性中它是作为一个对象处理的,如果直接绑定该属性名<%#Eval("Category") %>,将会显示如图 8-23 所示的结果。

书名	分类	☐全选
框架设计(第2版):CLR Via C#	BookShop.Models.Category	☐
C++程序设计教程(第二版)	BookShop.Models.Category	☐
深度探索 C++ 对象模型	BookShop.Models.Category	☐
Expert C# 2005 Business Objects中文版(第二版)	BookShop.Models.Category	☐
Visual C# 2005从入门到精通	BookShop.Models.Category	☐
12345678910...		

图 8-23　直接绑定分类属性的结果

显示的是分类的对象名。不过它也给了一个提示：此处作为对象进行处理。可以通过对象访问它的 Name 属性。

分类名称显示的代码如下：

```
<asp:TemplateField HeaderText = "分类">
  <EditItemTemplate>
    <asp:TextBox ID = "TextBox1" runat = "server" Text = '<%# Bind("Category.Name") %>'>
</asp:TextBox>
  </EditItemTemplate>
  <ItemTemplate>
    <asp:Label ID = "Label1" runat = "server" Text = '<%# Bind("Category.Name") %>'></asp:
Label>
  </ItemTemplate>
</asp:TemplateField>
```

通过这段代码，可以体会到面向对象编程的好处，如果使用传统的 DataSet 方式，只能通过联表查询才可以得到分类的名称。

使用 Bind()方法也可以实现在模板列中展示数据，在上述代码中，Bind()方法完全可以用 Eval()方法替代。

释疑：Bind()方法和 Eval()方法的区别。Eval()方法以数据字段的名称作为参数，通常用来在模板中绑定数据并以表达式的形式显示数据；针对使用数据源控件操作数据，是只读方法(单向数据绑定)。比如图书的 ISBN，并不想让用户做任何修改，就可以使用代码：<%# Eval("字段名").ToString().Trim() %>，Eval 还有一个重载方法，可以实现格式化，比如需要显示图书的出版日期，就可以使用：<%# Eval("PublishDate", "{0:dd/MM/yyyy}") %>，其中 0 代表对应的 PublishDate 字段，而 dd/MM/yyyy 指明了最终显示文本的格式。Bind()方法支持读/写功能(双向数据绑定)，该方法常常与数据控件(如 TextBox 控件)再加上数据源控件(如 ObjectDataSource 控件)一起使用，达到更新数据的目的，如图书的标题可以修改，可设置为：<%Bind("Title")%>，Bind()方法也支持 Eval()方法类似格式化重载方法。

(3) ButtonField 字段：这是一个按钮，可以通过 CommandName 设置按钮的命令，通常使用自定义代码实现命令按钮发生之后的操作。

(4) CommandField：CommandField 字段和 ButtonField 字段类似，它提供了创建命令按钮的功能。相比而言，它是一个特殊的字段，显示了用于在数据绑定控件中执行选择、编辑、插入或删除操作的命令按钮，自动生成命令，无须手写代码。

(5) HyperLinkField：HyperLinkField 允许将所绑定的数据以超链接的形式显示出来，可以定义绑定超链接的显示文字、超链接、打开窗口方式等。比如，图书需要显示一个指向详细页面的超链接，代码如下：

```
< asp: HyperLinkField  DataNavigateUrlFormatString  =  " DetailsView. aspx? id  =  { 0 }"
DataNavigateUrlFields = "Id" Text = "详细" />
```

(6) ImageField： ImageField 可以在 GridView 控件所呈现的表格中显示图片列，一般来说，它绑定的内容是图片的路径，比如图书的封面，就可以使用 ImageField：

```
<asp:ImageField DataImageUrlField="ISBN" DataImageUrlFormatString="~/images/BookCovers/{0}.jpg"
    HeaderText="封面">
</asp:ImageField>
```

(7) CheckBoxField：CheckBoxField 可以使用复选框的形式显示布尔类型的数据，注意，只有当该控件中有布尔型的数据时才可以使用 CheckBoxField。

4) 多选和光棒效果

多选和光棒效果都需要通过客户端脚本实现，但实现方式却差别很大。

(1) 多选效果：多选效果完全可以通过脚本进行实现。选择的控件最终生成一个 input 的 HTML 标签，它的类型是 checkbox。由于这个页面中只有一个组复选框，可以遍历所有 input 标签，对比其 type 属性是不是 checkbox，如果是 checkbox，就将它设置成选择状态。代码如下：

```
<script language="javascript">
    function GetAllCheckBox(CheckAll) {                    //以"全选"前面的复选框多项做参数
        var items = document.getElementsByTagName("input");  //获取所有 input 对象的列表
                                                              //并遍历该列表
        for (i = 0; i < items.length; i++) {
            if (items[i].type == "checkbox") {  //判断遍历 input 对象列表中的对象是否为 checkbox
                items[i].checked = CheckAll.checked;
            }
        }
    }
</script>
```

这段 JS 代码很容易理解。另外，还需要在"全选"前面的复选框上添加 onclick="GetAllCheckBox(this)"事件。如果在页面中有多组复选框，就稍微复杂一些，为每组复选框定义它们的 Name 属性，然后根据该属性确认哪组需要选中就行了。

(2) 光棒效果：实现光棒效果，就是改变某行的背景颜色。在样式表中，backgroundColor 属性用于设置对象的背景颜色。使用 JS 时，可以通过 this.backgroundColor 来获得或者设置某对象的背景色。

只需要给需要高亮显示的行添加两个事件 onmouseover 和 onmouseout。Onmouseover 在鼠标移动到对象所在区域时执行，而 onmouseout 在鼠标离开对象所在区域时执行。可以在这两个事件上编写代码，实现高亮效果。在表格中的每一行添加这两个时间就可以了。但表格中的行是绑定生成的，因此需要在这些生成的行上添加事件。

在 GridVies 中，有一个 RowDataBound 行绑定事件，可以在绑定行的时候，设置当前行鼠标移动效果，代码如下：

```
/// <summary>
/// gvBooks 的行绑定事件
/// </summary>
/// <param name="sender"></param>
```

```
/// <param name="e"></param>
protected void gvBooks_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        e.Row.Attributes.Add("onmouseover", "currentcolor = this.style.backgroundColor;
this.style.backgroundColor = '#6699ff'");
        // currentcolor 变量用于记录变色前的背景色,当鼠标离开时,背景色恢复到变色之前
        e.Row.Attributes.Add("onmouseout", "this.style.backgroundColor = currentcolor");
    }
}
```

按上述步骤完成后,运行 BookList.aspx 页面后的效果如图 8-24 所示。



图 8-24　编码指定数据源方式实现的图书列表页面

8.1.3　DataList 控件

在互联网广泛进入人类生活的今天,很多人都有在网上购物的体验,图 8-25 展示了亚马逊网上书店的图书列表页。

对于多行多列数据(或称为表格类数据)的展示,一般会选择前面讲过的 GridView 控件。对于模块化的单行多列或者多行单列的数据则通过 DataList 控件比较合适,如图 8-26 所示。

DataList 控件默认情况下以表格的形式显示数据,优点是用户可以为数据创建任意格式的布局。显示数据的格式在创建的模板中定义。可以为项、交替项、选定项和编辑项创建模板。表头、脚注和分隔符模板也用于自定义 DataList 的整体外观。通过在模板中添加 Button 和 LinkButton 等控件,可将列表项连接到代码,这些代码使用户得以在显示、选择



图 8-25　亚马逊网上书店图书列表

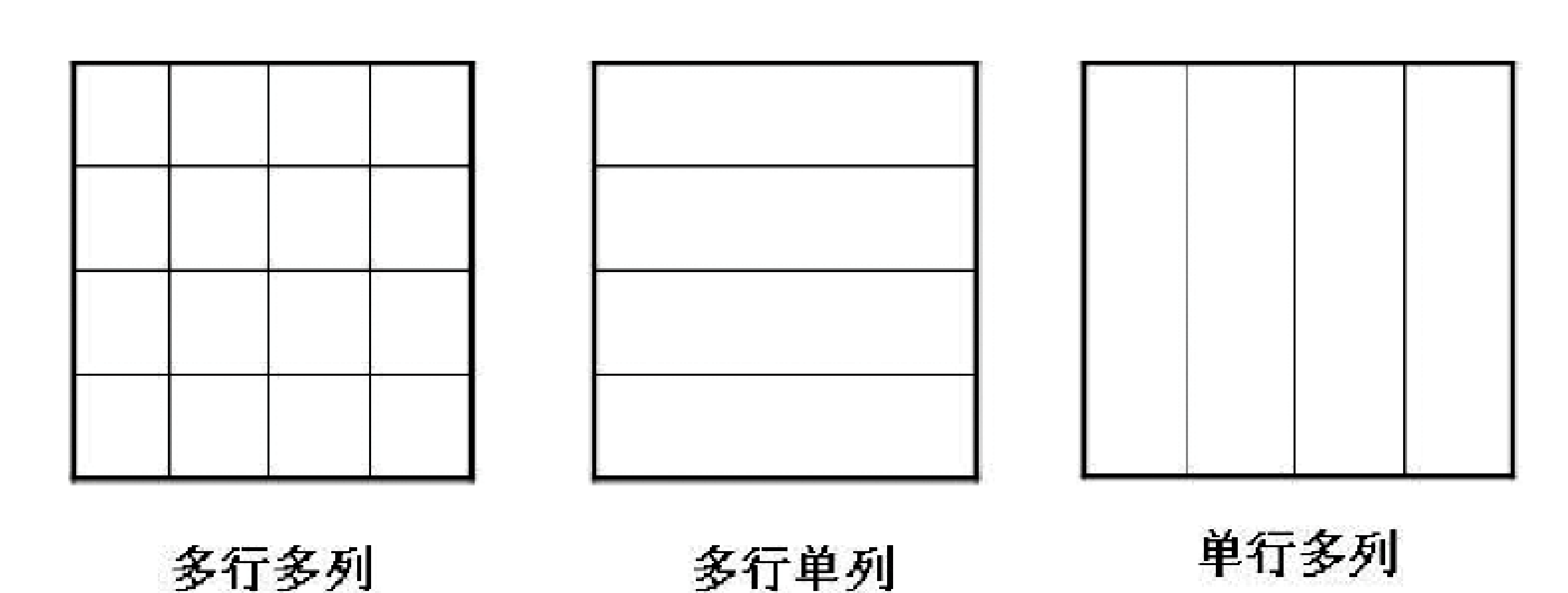


图 8-26　表格形式内容

和编辑模式之间进行切换,DataList 控件支持的模板如表 8-7 所示。

表 8-7　DataList 控件支持的模板

属　　性	描　　述
ItemTemplate	项模板,包含一些 HTML 元素和控件,将为数据源中的每一行呈现一次这些 HTML 元素和控件
HeaderTemplate	页眉模板,包含在列表的开始处呈现的文本和控件
FooterTemplate	页脚模板,包含在列表的结束处呈现的文本和控件
EditItemTemplate	编辑项模板,指定当某项处于编辑模式中时的布局。此模板通常包含一些编辑控件,如 TextBox 控件
SelectedItemTemplate	选中项模板,包含一些元素,当用户选择 DataList 控件中的某一项时将呈现这些元素
SeparatorTemplate	分隔符模板,包含在每项之间呈现的元素。典型的示例可能是一条直线(一般用<hr>)
AlternatingItemTemplate	交替项模板,包含一些 HTML 元素和控件,将为数据源中的每两行呈现一次这些 HTML 元素和控件

1. 使用 DataList 绑定数据

DataList 控件可用于创建模板化的列表数据，可以显示诸如多行单列的内容，可用于任何重复结构中的数据。

首先，将 DataList 控件拖入页面，DataList 控件不像 GridView 控件那样直接显示一个表格，而是需要编辑模板列。DataList 同 GridView 控件一样，可以通过右键快捷菜单选择"编辑模板"命令，进行模板的可视化编辑。DataList 中的模板与 GridView 相比，少了 InsertItemTemplate，增加了选定项模板(SelectItemTemplate)和分隔符模板(SeparatorItemTemplate)。下面以某电子商城的网上商品展示为例介绍 DataList 的使用，实现商品展示的页面效果如图 8-27 所示，电子商城系统的商品项表 Item 结构如表 8-8 所示。



图 8-27　电子商城的商品展示页面效果

表 8-8　电子商城的商品项表 Item

字　段　名	字段含义	数据类型	字段大小	备　　注
itemId	商品 Id	int		主键，自增 1
imagePath	商品图片路径	varchar	50	非空
itemname	商品名称	varchar	50	非空
saler	出售者	varchar	50	非空
telephone	电话	varchar	15	非空
price	商品价格	decimal	18	非空
categoryID	商品分类 Id	int		外键

列表中每行显示的内容样式是相同的，只是数据不同。因此每行显示的内容可以编写在一个ItemTemplate中，每行的布局可以通过一个3行3列的<Table>来实现。关键代码如下：

```
<asp:DataList ID="dlItems" runat="server" CssClass="margin" CellPadding="4" ForeColor
="#333333">
    <AlternatingItemStyle BackColor="White" />
    <FooterStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
    <HeaderStyle BackColor="#1C5E55" Font-Bold="True" ForeColor="White" />
    <HeaderTemplate>
    </HeaderTemplate>
    <ItemStyle BackColor="#E3EAEB" />
    <ItemTemplate>
        <table border="0" width="800" style="border-collapse: collapse">
            <tr>
                <td rowspan="3" style="width: 10%">
                    <asp:CheckBox ID="ch" runat="server" />
                    <asp:HiddenField runat="server" ID="hdid" Value='<%# Eval
("itemid") %>' />
                </td>
                <td rowspan="3" style="width: 20%">
                    <img alt="" src='<%# Eval("imagePath")%>' style="width: 100px;
height: 100px;" />
                </td>
                <td style="width: 50%"><%# Eval("itemname") %></td>
                <td rowspan="3" style="width: 20%"> <%# Eval("price") %></td>
            </tr>
            <tr>
                <td>联系电话: <%# Eval("telephone")%></td>
            </tr>
            <tr>
                <td>出售者: <%# Eval("saler")%></td>
            </tr>
        </table>
    </ItemTemplate>
    <SelectedItemStyle BackColor="#C5BBAF" Font-Bold="True" ForeColor="#333333" />
    <SeparatorTemplate>
        <hr />
    </SeparatorTemplate>
</asp:DataList>
```

接下来，在Default.aspx页面后置代码中完成数据绑定（数据访问层及业务逻辑层代码在教学资源中），DataList支持同GridView相同的绑定操作，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        this.BindList();
```

```
        }
    }
    public void BindList()
    {
        IList<Item> items = ItemService.GetItems();    //GetItems()为数据访问层定义的取得全
                                                       //部商品信息的方法
        this.dlItems.DataSource = items;
        this.dlItems.DataBind();
    }
```

运行使用 DataList 控件展示数据的页面,效果如图 8-27 所示,从源文件中可以发现数据其实是以 HTML 中的<Table>方式呈现的。

2. DataList 属性与事件

DataList 的常用属性和事件如表 8-9 所示。

表 8-9　DataList 的常用属性和事件

属性和事件		说　　明
属性	RepeatColumns	显示的列数。默认为 0,表示单行或单列显示
	RepeatDirection	获取或设置 DataList 的显示方向,Horizontal:水平,Vertical:垂直
	DataKeyField	获取或设置指定的数据源中的键字段
	DataKeys	获取每个记录的键值
事件	EditCommand	单击 DataList 中某项编辑按钮时引发
	ItemCommand	单击 DataList 中某项按钮时引发
	ItemDataBound	将 DataList 中的数据项绑定到数据时引发
	UpdateCommand	单击 DataList 中某项更新按钮时引发
	DeleteCommand	单击 DataList 中某项删除按钮时引发

1) RepeatColumns 与 RepeatDirection

RepeatColumns 与 RepeatDirection 属性为 DataList 控件的数据显示提供了更灵活的空间,效果如图 8-28 所示。



图 8-28　RepeatColumns 与 RepeatDirection 属性的商品展示效果(DataListProperty.aspx 页面)

可以将图 8-28 中的 DataList 看成单行,每行显示两列,代码如下:

```
<asp:DataList ID = "dlItems" runat = "server" CssClass = "margin" CellPadding = "4"
    ForeColor = "#333333"  Width = "500px"  RepeatColumns = "2"  RepeatDirection = "Horizontal">
    <% -- 省略其他代码 -- %>
</asp:DataList>
```

也可以将图 8-28 中的 DataList 看成是单列,每列显示两行,代码如下:

```
<asp:DataList ID = "dlItems" runat = "server" CssClass = "margin" CellPadding = "4"
    ForeColor = "#333333"  Width = "500px"  RepeatColumns = "2"  RepeatDirection = "Vertical">
    <% -- 省略其他代码 -- %>
</asp:DataList>
```

2) DataKeyField 与 DataKeys

如何在每项商品信息后添加一个“删除”按钮用于删除相应的商品,或添加一个“详细”按钮,单击后链接到商品的详细介绍页面。可以通过 DataKeys 属性获取 DataKeyField 属性设置的主键字段值。例如为图 8-28 中的每件商品添加一个“删除”按钮,如图 8-29 所示。



图 8-29 为每件商品信息添加一个“删除”按钮(DataListDelete.aspx 页面)

将 DataList 控件的 DataKeyField 属性设置成商品编号 itemId,单击“删除”按钮,在“删除”按钮引发的 ItemCommand 事件(或者 DeleteCommand 事件)中,通过 DataKeys 属性获取到主键值,执行删除操作,DataListDelete.aspx 页面后置代码如下:

```
protected void dlItems_ItemCommand(object source, DataListCommandEventArgs e)
{
    int index = e.Item.ItemIndex;
    int curId = (int)this.dlItems.DataKeys[index];
    //执行删除操作
    try
    {
        ItemService.DeleteItem(curId);   //DeleteItem(curId)为数据访问层定义的删除商品信
                                          //息的方法
        BindList();                       //执行删除操作后,重新绑定
    }
    catch (Exception ex)
```

```
        {
            ScriptManager.RegisterClientScriptBlock(this, typeof(Page), "showerror", "alert('删
除过程中出现错误!" + ex.Message + "')", true);
        }
    }
```

3. DataList 排序

在网上购物时,用户通常喜欢按照自己关注的参数排序查看商品。比如购买内存条时,需要提供给用户按内存大小排序的功能。DataList 不像 GridView 控件具有内置的排序和分页功能,需要手工编码实现。下面通过为电子商城的商品展示添加排序功能来了解 DataList 如何实现编码实现排序,添加的功能如图 8-30 所示。



图 8-30　电子商城商品展示的排序效果(DataListSort.aspx 页面)

分析:实现排序,最直接的办法是在数据库端就排好序,所以在数据访问层直接添加排序条件。将数据绑定到页面,需要通过业务逻辑层传递相关参数(这里为排序条件)。

首先在 DataList 网站项目下的 DataListSort.aspx 页面中添加一个 DropDownList 供用户选择,代码如下:

```
<asp:DropDownList runat="server" ID="ddlSort" AutoPostBack="True"
                  OnSelectedIndexChanged="ddlSort_SelectedIndexChanged">
    <asp:ListItem Text="按类别排序"></asp:ListItem>
    <asp:ListItem Text="按价格降序"></asp:ListItem>
    <asp:ListItem Text="按价格升序"></asp:ListItem>
</asp:DropDownList>
```

其次，通过 DropDownList 下拉列表选项的改变获取排序参数。表示层的代码如下：

```
/// <summary>
/// DropDownList 控件的 SelectedIndexChanged 事件方法代码
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void ddlSort_SelectedIndexChanged(object sender, EventArgs e)
{
    string strName = GetSortNameByValue();
    this.dlItems.DataSource = ItemService.GetItemsByOrderStr(strName);
    this.dlItems.DataBind();
}
/// <summary>
/// 根据用户选择确定排序条件
/// </summary>
/// <returns></returns>
private string GetSortNameByValue()
{
    string name = string.Empty;
    string strValue = this.ddlSort.SelectedIndex.ToString();
    switch (strValue)
    {
        case "0": name = "categoryID"; break;
        case "1": name = "price desc"; break;
        case "2": name = "price asc"; break;
        default: break;
    }
    return name;
}
```

数据访问层的代码如下：

```
public static IList<Item> GetItemsByOrderStr(string strOrderField)
{
    IList<Item> items = new List<Item>();
    string sql = "select * from item order by " + strOrderField;
    SqlDataReader dr = SqlHelper.ExecuteReader(SqlHelper.ConnectionString,
          CommandType.Text, sql, null);
    while (dr.Read())
    {
```

```
            Item item = new Item();
            item.ItemID = Int32.Parse(dr["itemID"].ToString());
            item.ImagePath = dr["imagePath"].ToString().Replace("~/", "");
            item.ItemName = dr["itemName"].ToString();
            item.Saler = dr["saler"].ToString();
            item.Telephone = dr["telephone"].ToString();
            item.Price = Convert.ToDecimal(dr["price"]);
            items.Add(item);
        }
        dr.Close();
        return items;
    }
```

注意：数据访问层的排序方法有一个排序字段参数。如果是多个字段，可以在 SQL 语句中用逗号把它们隔开。

4. DataList 分页

大多数电子商务网站的商品展示都需要分页功能，下面仍以电子商城的商品展示为例介绍如何实现如图 8-31 所示的分页功能。



图 8-31　电子商城商品展示的分页效果(DataListPageDemo.aspx 页面)

如何实现每页显示两条商品记录，提供“第 X 页，共 X 页”的信息提示，并实现单击“上一页”按钮和“下一页”按钮的翻页功能。

分页的方法很多，下面基于 PagedDataSource 类来阐述分页功能的实现。

PagedDataSource 是.NET 内置的分页类，它封装了数据绑定控件与分页相关的属性，只要为它指定数据源、当前页和 PageSize，即可自动计算其他相关属性，常用属性如表 8-10 所示。

表 8-10 PagedDataSource 类的常见属性

属性名称	说　明
CurrentPageIndex	当前页,起始页为 0
PageCount	总页数
Count	总记录数
PageSize	每页记录数
AllowPaging	控件是否实现自动分页功能
DataSource	数据源

每次单击“上一页”或“下一页”按钮会造成页面回传,在于服务器交互中需要保持当前页数、排序条件。因为该分页和排序信息仅需要在当前页面有效,所以前面学过的 Session、Cookie、Application 状态保持方式并不合适。在前面探讨过页面级的状态保持:ViewState。使用页面级状态保持的好处就是不影响其他页面的分页,ViewState 的语法和 Session 一样,如下所示:

```
ViewState["名称"] = 值
```

或者

```
ViewState.Add("名称",值)
```

例如,要把页数“0”信息存入 ViewState,就可以写成如下所示代码:

```
ViewState["Page"] = 0;
```

或者

```
ViewState.Add("Page",0);
```

ViewState 状态保持方式的本质是在页面上放置一个隐藏域:

```
< input type = "hidden" name = "__VIEWSTATE" value = "">
```

每次数据回传,该隐藏域的内容也一起回传,从而进行状态信息的保持。与<asp:HiddenField></HiddenField>不同的是,ViewState 可以存储对象,而 HiddenField 只能存放字符串。

本示例采用 ViewState 定义两个属性,用来保存当前页和总页数,代码如下:

```
/// < summary >
/// 当前页
/// </summary >
public int CurrentPageIndex
{
    set
    {
        ViewState["CurrentPageIndex"] = value;
```

```
        }
        get
        {
            return Convert.ToInt32(ViewState["CurrentPageIndex"]);
        }
    }
    /// <summary>
    /// 总页数
    /// </summary>
    public int PageCount
    {
        set
        {
            ViewState["PageCount"] = value;
        }
        get
        {
            return Convert.ToInt32(ViewState["PageCount"]);
        }
    }
```

在页面 DataListPageDemo.aspx 中拖放两个 Button 和一个 Label 控件，代码如下：

```
<asp:Label ID="lblInfo" runat="server" Text="Label"></asp:Label>
<asp:Button ID="btnPre" runat="server" Text="上一页" OnClick="BtnPre_Click" />
<asp:Button ID="btnNext" runat="server" Text="下一页" OnClick="BtnNext_Click" />
```

当单击“上一页”或“下一页”按钮实现翻页功能，DataListPageDemo.aspx 页面后置代码如下：

```
/// <summary>
/// "上一页"按钮事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void BtnPre_Click(object sender, EventArgs e)
{
    this.CurrentPageIndex--;
    this.BindList();
    this.SetButtonStatus();
}
/// <summary>
/// "下一页"按钮事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void BtnNext_Click(object sender, EventArgs e)
{
    this.CurrentPageIndex++;
```

```
        this.BindList();
        this.SetButtonStatus();
    }
```

其中，SetButtonStatus()方法用于设置按钮的可用性，单击“上一页”或“下一页”按钮实现翻页功能时，如果已经是第一页，则“上一页”按钮为灰色，如果已经是最后一页，则“下一页”按钮为灰色，代码如下：

```
/// <summary>
/// 设置按钮的可用性
/// </summary>
private void SetButtonStatus()
{
    //如果当前页是最后一页
    if (CurrentPageIndex >= this.PageCount)
    {
        this.btnNext.Enabled = false;
    }
    else
    {
        this.btnNext.Enabled = true;
    }
    if (CurrentPageIndex <= 1)
    {
        this.btnPre.Enabled = false;
    }
    else
    {
        this.btnPre.Enabled = true;
    }
}
```

注意：不要在“上一页”、“下一页”按钮的单击事件内控制按钮的可用性，而是写在一个方法内。如果页面上还有“首页”、“末页”等按钮，那么把控制方法写在按钮单击事件方法内是很容易出错的。

页面 DataListPageDemo.aspx 的 Page_Load 的事件方法代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        CurrentPageIndex = 1;
        this.BindList();
        SetButtonStatus();
    }
}
```

其中，BindList()方法实现基于 PagedDataSource 类的商品展示分页功能，代码如下：

```
public void BindList()
{
    PagedDataSource pds = new PagedDataSource();  //定义一个 PagedDataSource 实例
    pds.AllowPaging = true;                       //设置数据绑定控件启用分页
    pds.CurrentPageIndex = CurrentPageIndex - 1;  //使用状态保持保存当前页数
    pds.DataSource = ItemService.GetItems();      //指定 PagedDataSource 数据源
    pds.PageSize = this.pageSize;                 //设置每页记录数
    this.PageCount = pds.PageCount;
    this.lblInfo.Text = "第" + CurrentPageIndex + "页,共" + this.PageCount + "页";
    this.dlItems.DataSource = pds;  //将 DataList 的数据源设置成 PagedDataSource
    this.dlItems.DataBind();
}
```

可以看出，用 PagedDataSource 分页比较方便，使用过程如下：

（1）指定 PagedDataSource 实例对象的数据源为 GetItems()方法返回的数据集合，需要注意的是，PagedDataSource 对象的数据源不能是 DataTable 或 DataSet 对象。

（2）分别设置允许分页(AllowPaging)、页面大小(PageSize)、当前页(CurrentPageIndex)等属性的值。

（3）指定数据绑定控件的数据源为该实例对象，并绑定。

8.1.4　Repeater 控件

1. Repeater 概述

DataList 控件灵活性好，能够自动生成“<Table>”中的“<TR>”、“<TD>”等相关标签，不过在 DIV＋CSS 网页布局越来越流行的今天，美工提供了苛刻的设计页，没有 Table，样式写在 CSS 中，要求生成页面的标签完全套用模板页面，精确显示，比如生成如下 HTML 代码：

```
<ul class = "title_ul2">
    <li class = "title_booklist0" ><a href = "BookDetail.aspx?id = 4961">数据结构(C 语言版)
(第二版)</a></li>
    <li class = "title_booklist1">肖宏启  等著…… </li>
    <li class = "title_booklist2">电子工业出版社 </li>
    <li class = "title_booklist3"> 2014 - 1 - 10 </li>
    <li class = "title_booklist4">￥43.00 </li>
</ul>
```

如果不生成其他的代码，可以考虑使用 Repeater 控件，Repeater 控件是一个数据绑定容器控件，可以从页的任何可用数据中创建出自定义列表。Repeater 控件不具备内置的布局或样式，用户必须通过创建模板为 Repeater 控件提供布局。当该页运行时，Repeater 控件依次为通过数据源中的每个记录重复此布局。由于 Repeater 控件没有默认的外观，因此可以使用该控件创建多种列表，包括：

- 表格布局。
- 逗号分隔的列表(例如,a、b、c、d 等)。
- XML 格式的列表。

2. 使用 Repeater 绑定数据

Repeater 控件专门用于精确内容的显示,也是基于模板的方式,其常用模板属性如表 8-11 所示,但 Repeater 不会自动生成任何用于布局的代码,甚至没有一个默认的外观,它完全是通过模板来控制,而且也只能通过源代码视图进行模板的编辑。从表 8-11 可知,与 DataList 相比, Repeater 可以模板更少,没有编辑和选择模板,由于不能自动生成任何 HTML 标签,所以带来了效率上的提升,也使精确展示数据成为可能。

表 8-11 Repeater 控件支持的模板

属　性	描　述
ItemTemplate	项模板,包含要为数据源中每个数据项都要呈现一次的 HTML 元素和控件
HeaderTemplate	页眉模板,包含在列表的开始处呈现的文本和控件
FooterTemplate	页脚模板,包含在列表的结束处呈现的文本和控件
SeparatorTemplate	分隔符模板,包含在每项之间呈现的元素。典型的示例可能是一条直线(分隔符)
AlternatingItemTemplate	交替项模板,包含一些 HTML 元素和控件,将为数据源中的每两行呈现一次这些 HTML 元素和控件

用 Repeater 实现新知书店图书检索页页面,如图 8-32 所示。



图 8-32 新知书店图书检索效果(BookSearchList.aspx 页面)

这里需要注意的是,由于 Repeater 不提供任何布局的内容,所以在项模板中要将用于布局的标签写完整,比如＜ul＞。

另外,标题头的内容可以写在标题头模板(页眉模板)中,也可以写在 Repeater 控件外。两种方式的效果是一致的。检索页与图书列表页面功能上相似,除了调用的方法之外,其他

后台代码几乎完全一样，页面(BookSearchList.aspx)代码如下所示：

```
<div class = "main_booklist">
 <dl>
   <dt>
    <asp:Repeater ID = "rpBookList" runat = "server">
        <HeaderTemplate>                //页眉项模板(标头项模板)
          <ul class = "title_ul2">
            <li class = "title_booklist0">书名</a></li>
            <li class = "title_booklist1">作者</li>
            <li class = "title_booklist2">出版社</li>
            <li class = "title_booklist3">出版时间</li>
            <li class = "title_booklist4">单价</li>
          </ul>
        </HeaderTemplate>
        <ItemTemplate>                  //内容项模板
          <ul>
              <li class = "title_booklist0"><a href = "BookDetail.aspx?bid = <%# Eval
("Id") %>" target = "_blank">
                 <%# Eval("Title") %></a></li>
            <li class = "title_booklist1"><%# Eval("Author") %></li>
            <li class = "title_booklist2"><%# Eval("publisher.Name") %></li>
            <li class = "title_booklist3"><%# Eval("publishDate","{0:yyyy - mm - dd}") %>
</li>
            <li class = "title_booklist4"><%# Eval("UnitPrice","{0:C}") %></li>
          </ul>
        </ItemTemplate>
        <AlternatingItemTemplate>       //交替项模板
          <ul class = "title_ul2" style = "clear: both">
              <li class = "title_booklist0"><a href = "BookDetail.aspx?bid = <%# Eval
("Id") %>" target = "_blank">
                 <%# Eval("Title") %></a></li>
            <li class = "title_booklist1"><%# Eval("Author") %></li>
            <li class = "title_booklist2"><%# Eval("publisher.Name") %></li>
            <li class = "title_booklist3"><%# Eval("publishDate","{0:yyyy - mm - dd}") %>
</li>
            <li class = "title_booklist4"><%# Eval("UnitPrice","{0:C}") %></li>
          </ul>
        </AlternatingItemTemplate>
        <FooterTemplate>
        </FooterTemplate>
    </asp:Repeater>
    </dt>
  </dl>
</div>
```

8.1.5　其他数据绑定控件

至此，已经详细讲解了复杂数据绑定控件 GridView、DataList、Repeater 的使用，其实，

它们的使用和要注意的问题大同小异，除了上述控件之外，ASP.NET 还提供了 DetailsView、FormView、ListView 和 DataPager 控件，在此不再详细阐述它们的使用，只做简单介绍，在后续章节结合单元任务的实施再演示它们的使用。

1. DetailsView 控件

DetailsView 控件可以一次显示一个数据记录。当需要深入研究数据库文件中的某一个记录时，DetailsView 控件就可以大显身手。DetailsView 经常在主控/详细方案中与 GridView 控件配合使用。用户使用 GridView 控件来选择列，用 DetailsView 来显示相关的数据。

DetailsView 控件依赖于数据源控件的功能执行诸如更新、插入和删除记录等任务。DetailsView 控件不支持排序。DetailsView 控件可以自动对其关联数据源中的数据进行分页，但前提是数据由支持 ICollection 接口的对象表示或基础数据源支持分页。DetailsView 控件提供用于在数据记录之间导航的用户界面(UI)。若要启用分页行为，要将 AllowPaging 属性设置为 true。多数情况下，上述操作的实现无须编写代码。

DetailView 有一个 DefaultMode 属性，控制默认的显示模式，该属性有 3 个可选值。

- DetailsViewMode.Edit：编辑模式，用户可以更新记录的值。
- DetailsViewMode.Insert：插入模式，用户可以向数据源中添加新记录。
- DetailsViewMode.ReadOnly：只读模式，这是默认的显示模式。

2. FormView 控件

FormView 控件提供了内置的数据处理功能，只需绑定到支持这些功能的数据源控件，并进行配置，无须编写代码就可以实现对数据的分页和增删改功能。

要使用 FormView 内置的增删改功能需要为更新操作提供 EditItemTemplate 和 InsertItemTemplate 模板，FormView 控件显示指定的模板以提供允许用户修改记录内容的用户界面。

FormView 控件的各个项通过自定义模板来呈现，控件并不提供内置的实现某一功能(如删除)的特殊按钮类型，而是通过按钮控件的 CommandName 属性与内置的命令相关联。FormView 提供如下命令类型(区分大小写)：

- Edit——引发此命令控件转换到编辑模式，并用已定义的 EditItemTemplate 呈现数据。
- New——引发此命令控件转换到插入模式，并用已定义的 InsertItemTemplate 呈现数据。
- Update——此命令将使用用户在 EditItemTemplate 界面中的输入值在数据源中更新当前所显示的记录。引发 ItemUpdating 和 ItemUpdated 事件。
- Insert——此命令用于将用户在 InsertItemTemplate 界面中的输入值在数据源中插入一条新的记录。引发 ItemInserting 和 ItemInserted 事件。
- Delete——此命令删除当前显示的记录。引发 ItemDeleting 和 ItemDeleted 事件。
- Cancel——在更新或插入操作中取消操作和放弃用户输入值，然后控件会自动转换到 DefaultMode 属性指定的模式。

3. ListView 控件

ListView 控件是 ASP.NET 3.5 提供的新控件，它很好地集成了 GridView、DataList 和 Repeater 的优点。类似于 GridView，它支持数据编辑、删除和分页；类似于 DataList，它支持多列和多行布局；类似于 Repeater，它允许完全控制控件生成的标记。ListView 通过模板显示和管理数据，可用模板及常用事件如表 8-12 所示。

表 8-12　ListView 控件支持的模板和事件

模板和事件		描　　述
模板	<LayoutTemplate>	控件的容器。它使得可定义一个放置单独数据项(像 Reviews)的位置，然后通过 ItemTemplate 和 AlternatingItemTemplate 表示的数据项作为容器的子元素添加。它还可能包含一个 DataPager 对象
	<ItemTemplate> <AlternatingItemTemplate>	项模板、交替项模板，定义控件的只读模式。当一起使用时，奇偶行有着不同的外观(通常是不同的背景色)
	<SelectedItemTemplate>	允许定义当前活动或选择项的外观
	<InsertItemTemplate> <EditItemTemplate>	这两个模板允许定义用于插入和更新列表中的项的用户界面。通常，放置文本框、下拉列表和其他服务器控件等到这些模板中，将它们与底层数据源绑定
	<ItemSeparatorTemplate>	定义放置在列表中项之间的标记。可用于在项之间添加线、图像或其他标记
	<EmptyDataTemplate>	在控件无数据显示时显示。可以添加文本或其他标记，告诉用户无数据显示
事件	AfterLabelEdit	在编辑了标签后，引发该事件
	BeforeLabelEdit	在用户开始编辑标签前，引发该事件
	ColumnClick	在单击一个列时，引发该事件
	ItemActivate	在激活一个选项时，引发该事件

ListView 中至少必须包含两个模板：LayoutTemplate 和 ItemTemplate。LayoutTemplate 模板是 ListView 用来显示数据的布局模板，ItemTemplate 则是每一条数据的显示模板，将 ItemTemplate 模板放置在 LayoutTemplate 模板中可以实现定制的布局。

4. DataPager 控件

ASP.NET 3.5 提供的新控件。在 ASP.NET 的前几个版本中，分页只是通过一些控件(如 GridView 和 DetailsView)内置的功能实现，或是通过手动编写代码实现(基于存储过程的分页，由于比较复杂，本书没有阐述，有兴趣的读者可自行查找资料)。

DataPager 是个单独的控件，可用它来扩展另一个数据绑定控件。目前，只能使用 DataPager 为 ListView 控件提供分页功能，将 DataPager 与 ListView 控件关联后，分页将自动完成。将 DataPager 与 ListView 控件关联有两种方法：

(1) 在 ListView 控件的 LayoutTemplate 模板中定义它。此时，DataPager 将明确它将给哪个控件提供分页功能。

(2) 在 ListView 控件外部定义它。需要将 DataPager 的 PagedControlID 属性设置为有效 ListView 控件的 ID。如果想将 DataPager 控件放到页面不同的地方，例如 Footer 或

SideBar 区域，也可以在 ListView 控件的外部进行定义。

DataPager 控件包括两种样式：一种是“文本”样式，第二种是“数字”样式，如图 8-33 所示。

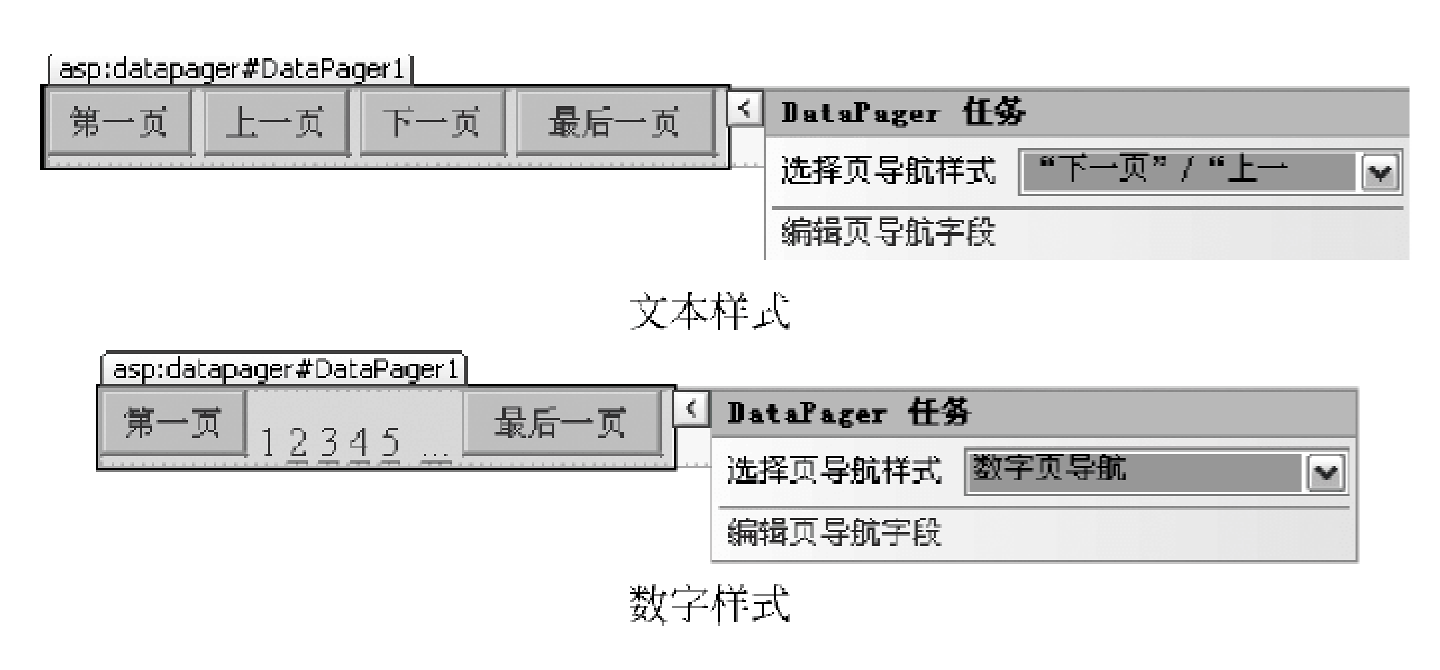


图 8-33 DataPager 控件的样式

当使用“文本”样式时，DataPager 控件的 HTML 实现代码如下所示。

```
<asp:DataPager ID = "DataPager1" runat = "server">
  <Fields>
    <asp:NextPreviousPagerField ButtonType = "Button" ShowFirstPageButton = "True"
        ShowLastPageButton = "True" />
    </Fields>
</asp:DataPager>
```

当使用“数字”样式时，DataPager 控件的 HTML 实现代码如下所示。

```
<asp:DataPager ID = "DataPager1" runat = "server">
    <Fields>
      <asp:NextPreviousPagerField ButtonType = "Button" ShowFirstPageButton = "True"
            ShowNextPageButton = "False" ShowPreviousPageButton = "False" />
      <asp:NumericPagerField />
      <asp:NextPreviousPagerField ButtonType = "Button" ShowLastPageButton = "True"
            ShowNextPageButton = "False" ShowPreviousPageButton = "False" />
    </Fields>
</asp:DataPager>
```

除了可以通过默认的方法来显示分页样式意外，还可以通过向 DataPager 中的 Fields 中添加 TemplatePagerField 的方法来自定义分页样式。在 TemplatePagerField 中添加 PagerTemplate，在 PagerTemplate 中添加任何服务器控件，这些服务器控件可以通过实现 TemplatePagerField 的 OnPagerCommand 事件来实现自定义分页。

8.2 单元任务

任务 8-2-1 实现“新知书店”管理员端的图书查询页面

【任务描述】

实现如图 8-34 所示的管理员端的图书查询页面，具体要求如下：

- 后台图书列表页面加载默认显示全部图书的书名、作者和类别信息。
- 实现 DropDownList 中提供用户书名、内容简介、出版社和作者的关键字查询功能。
- 实现光棒效果、分页效果。
- 当单击图书名称时，链接至图书编辑页面 BookDetail.aspx。



图 8-34　“新知书店”管理员端图书查询页面

【任务实施】

1. 图书查询数据访问层与业务逻辑层的实现

当管理员登录后，进入图书列表页时，默认显示全部图书的相关信息，当从下拉列表控件 DropDownList 选择检索类别，输入查询关键字并单击“查询”按钮后，图书列表页面重新显示符合条件查询的图书。

1）图书查询数据访问层的实现

在 BookShopDAL 项目的 BookService.cs 类中，添加一个获取所有图书信息列表对象的方法 GetBooks()，添加方法 GetBooks(BookQueryCategories category, string keyWord)，根据检索类别和关键字获取图书信息列表对象，代码如下：

```
⋮
using BookShop.Models;
using System.Configuration;
namespace BookShop.DAL
{
  public class BookService
  {
    string connection = ConfigurationManager.ConnectionStrings["BookShop"].ConnectionString;
    public List<Book> GetBooks()
    {
        string sqlAll = "SELECT * FROM Books";
        return GetBooksBySql(sqlAll);
    }
    public List<Book> GetBooksBySql(string safeSql)
```

```
    {
        //代码在示例 8 - 2 的表 8 - 4 中,此处省略
    }

    /// <summary>
    /// 获取书籍列表
    /// </summary>
    /// <param name="category">查询类别</param>
    /// <param name="keyWord">关键字</param>
    /// <returns>泛型书籍集合</returns>
    public List<Book> GetBooks(BookQueryCategories category, string keyWord)
    {
        List<Book> list = new List<Book>();
        SqlParameter[] para = new SqlParameter[]
        {
            new SqlParameter("@QueryCategory", category.ToString()),
            new SqlParameter("@KeyWord", keyWord)
        };
        //根据存储过程 sp_QueryBooks 获取 DataSet
        DataSet ds = SqlHelper.ExecuteDataset(this.connection, CommandType.StoredProcedure, "sp_
QueryBooks", para);
        if (ds.Tables.Count > 0)
        {
            DataTable dt = ds.Tables[0];
            foreach (DataRow row in dt.Rows)
            {
                Book book = new Book();
                book.Id = (int)row["Id"];
                book.Title = (string)row["Title"];
                book.Author = (string)row["Author"];
                book.PublishDate = (DateTime)row["PublishDate"];
                book.ISBN = (string)row["ISBN"];
                book.UnitPrice = (decimal)row["UnitPrice"];
                book.ContentDescription = (string)row["ContentDescription"];
                book.TOC = (string)row["TOC"];
                book.Clicks = (int)row["Clicks"];
                book.Publisher = new PublisherService().GetPublisherById((int)row
["PublisherId"]);      //FK
                book.Category = new CategoryService().GetCategoryById((int)row
["CategoryId"]);      //FK
                list.Add(book);
            }
        }
      return list;
    }
  }
}
```

代码中的存储过程 sp_QueryBooks 实现数据的查询功能，存储过程代码如下：

```
/ * 根据类别和关键字查询书籍 Created in 20140220 * /
ALTER PROC [dbo].[sp_QueryBooks]
@QueryCategory NVARCHAR(10),         -- 查询类型
@KeyWord NVARCHAR(50)                -- 查询关键字
AS
IF(@QueryCategory = '书名')
SELECT  *  FROM Books WHERE Title LIKE '%' + @KeyWord + '%'
ELSE IF(@QueryCategory = '内容简介')
SELECT  *  FROM Books WHERE ContentDescription LIKE '%' + @KeyWord + '%'
ELSE IF(@QueryCategory = '作者')
SELECT  *  FROM Books WHERE Author LIKE '%' + @KeyWord + '%'
ELSE IF(@QueryCategory = '出版社')
SELECT Books. *  FROM Books INNER JOIN Publishers
ON Books.PublisherId = Publishers.Id
WHERE Publishers.Name LIKE  '%' + @KeyWord + '%'
```

在 BookShopDAL 项目的 CategoryService.cs 类中，添加一个查询所有图书类别的所有字段信息的列表对象的 GetSelectByCategory()方法，代码如下：

```
⋮
using BookShop.Models;
using System.Configuration;
namespace BookShop.DAL
{
    public class SelectByCategoryService
    {
            string  connection  =  ConfigurationManager. ConnectionStrings [ " BookShop "].
ConnectionString;
        public List<SelectByCategory> GetSelectByCategory()
        {
            List<SelectByCategory> SelectByCategoryList  =  new List<SelectByCategory>();
            string sql  =  "SELECT  *  FROM SelectByCategory";
            using (SqlDataReader reader  =  SqlHelper.ExecuteReader(this.connection,
                                                      CommandType.Text,  sql))
            {
                while(reader.Read())
                {
                    SelectByCategory selectbycategory  =  new SelectByCategory();
                    selectbycategory.Id  =  (int)reader["Id"];
                    selectbycategory.Name  =  (string)reader["Name"];
                    SelectByCategoryList.Add(selectbycategory);
                }
            }
            return SelectByCategoryList;
        }
    }
}
```

2）图书查询业务逻辑层的实现

在 BookShopBLL 项目中新建类文件 BookManager.cs，实现图书查询业务逻辑功能，代码如下：

```
⋮
using BookShop.DAL;
using BookShop.Models;
namespace BookShop.BLL
{
  public class BookManager
  {
    public List<Book> GetBooks()
    {
        return new BookService().GetBooks();
    }
    public List<Book> GetBooks(BookQueryCategories category, string keyWord)
    {
        return new BookService().GetBooks(category, keyWord);
    }
  }
}
```

2. 图书查询表示层的实现

1）表示层页面设计

在 Web 站点项目的文件夹 Admin 下，根据后台母版页 Admin.master 新建图书列表内容页 BookList.aspx，并从工具箱将 DropDownList 控件拖入到面中并修改其 ID 属性为 ddlQueryCategories，代码如下：

```
<table>
  <tr>
    <td>   检索类别:
        <asp:DropDownList ID="ddlQueryCategories" runat="server">
            <asp:ListItem>书名</asp:ListItem>
            <asp:ListItem>内容简介</asp:ListItem>
            <asp:ListItem>出版社</asp:ListItem>
            <asp:ListItem>作者</asp:ListItem>
        </asp:DropDownList>
        关键字:
        <asp:TextBox ID="txtKeyWord" runat="server"></asp:TextBox>
        <asp:Button ID="btnQuery" runat="server" Text="查询" OnClick="btnQuery_Click" />
    </td>
    <td style="width: 100px;">   
        <asp:HyperLink ID="hlkAddBook" runat="server" NavigateUrl="~/Admin/BookEdit.
aspx">添加书籍</asp:HyperLink>
    </td>
  </tr>
</table>
```

从工具箱将一个 GridView 控件拖入到 BookList. aspx 页面中，修改其 ID 属性为 gvBooks，设置 DataKeyNames 属性值为图书表 Books 的关键字 id，设置分页相关的 AllowPaging、Mode、FirstPageText、LastPageText、NextPageText、PreviousPageText 等属性值，代码如下：

```
<asp:GridView runat="server" ID="gvBooks" AutoGenerateColumns="False" AllowSorting=
"true"
        CellPadding="0" CellSpacing="0" CssClass="data_table"
        OnRowDataBound="gvBooks_RowDataBound"
        OnPageIndexChanging="gvBooks_PageIndexChanging" AllowPaging="True"
        DataKeyNames="id"  OnSorting="gvBooks_Sorting" Width="740px" PageSize="3">
    <Columns>
            <asp:TemplateField Visible="False">
                <ItemTemplate>
                    <asp:Label ID="lblId" runat="server" Text='<%# Bind("Id") %>'></asp:
Label>
                </ItemTemplate>
            </asp:TemplateField>
            <asp:HyperLinkField DataNavigateUrlFormatString="BookDetail.aspx?id={0}"
                HeaderText="书名" ControlStyle-Width="180" Text="详细"
                    DataNavigateUrlFields="id" DataTextField="Title"  SortExpression=
"Title">
                <ControlStyle Width="180px"></ControlStyle>
            </asp:HyperLinkField>
            <asp:BoundField DataField="Author" HeaderText="作者" />
            <asp:TemplateField HeaderText="类别">
                <ItemTemplate>
                    <asp:Label ID="lblCategory" runat="server" Text='<%# Eval("Category.
Name") %>'>
                    </asp:Label>
                </ItemTemplate>
            </asp:TemplateField>
    </Columns>
    <PagerSettings FirstPageText="首页" LastPageText="最后一页" Mode="NextPreviousFirstLast"
        NextPageText="下一页" PreviousPageText="上一页" />
    <PagerStyle CssClass="pages" />
    <RowStyle BackColor="#DDF5D9" />
    <SelectedRowStyle BackColor="#CE5D5A" Font-Bold="True" ForeColor="White" />
    <AlternatingRowStyle BackColor="White" />
</asp:GridView>
```

2）编程实现图书查询相关功能

在图书列表页的后置代码文件 BookList. aspx. cs 中编程实现图书查询功能，代码如下：

```
⋮
using BookShop.BLL;
using BookShop.Models;
```

```
public partial class Admin_BookList : System.Web.UI.Page
{
    /// <summary>
    /// 页面加载事件方法
    /// </summary>
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            this.gvBooks.DataSource = new BookManager().GetBooks();
            this.gvBooks.DataBind();
        }
    }

    /// <summary>
    /// 类别绑定方法
    /// </summary>
    protected void BindSelectByCategory()
    {
        this.ddlQueryCategories.DataSource = new SelectByCategoryManager().GetSelectByCategory();
        this.ddlQueryCategories.DataTextField = "Name";   //用于显示的字段
        this.ddlQueryCategories.DataValueField = "Id";    //用于存值的字段
        this.ddlQueryCategories.DataBind();
        this.ddlQueryCategories.Items.Insert(0, new ListItem("===请选择===", "0"));
    }

    /// <summary>
    /// gvBooks控件页索引改变事件
    /// </summary>
    protected void gvBooks_PageIndexChanging(object sender, GridViewPageEventArgs e)
    {
        this.gvBooks.PageIndex = e.NewPageIndex;
        this.gvBooks.DataSource = new BookManager().GetBooks();
        this.gvBooks.DataBind();
    }

    /// <summary>
    /// 对行进行了数据绑定后的事件方法,实现光棒效果
    /// </summary>
    protected void gvBooks_RowDataBound(object sender, GridViewRowEventArgs e)
    {
        if (e.Row.RowType == DataControlRowType.DataRow)
        {
            e.Row.Attributes.Add("onmouseover","currentcolor=this.style.backgroundColor;
this.style.backgroundColor='#6699ff'");
            e.Row.Attributes.Add("onmouseout", "this.style.backgroundColor=currentcolor");
        }
    }

    /// <summary>
```

```
    /// 单击"查询"按钮事件方法
    /// </summary>
    protected void btnQuery_Click(object sender, EventArgs e)
    {
        //将一个或多个枚举常数的名称或数字值的字符串表示转换成等效的枚举对象
        BookQueryCategories category = (BookQueryCategories) Enum. Parse (typeof
(BookQueryCategories), ddlQueryCategories.SelectedValue);
        this.gvBooks.DataSource = new BookManager().GetBooks(category, txtKeyWord.Text);
        this.gvBooks.DataBind();
    }
}
```

任务 8-2-2　实现“新知书店”管理员端的图书详细信息的更新功能

【任务描述】

在管理端的图书信息列表页面中，单击某一条图书记录对应的“编辑”按钮，如图 8-35 所示，进入当前记录的图书详细信息页面，并显示当前图书记录的详细信息，如图 8-37 所示，在图书详细信息页面中，单击“保存”按钮，将修改后的当前记录的图书详细信息更新到数据库中，修改成功后，给出提示，并将页面跳转到图书信息列表页面，要求：

- 除出版社、分类、内容摘要外都需要进行非空验证。
- 图书的图片需要实现上传，更新功能，只允许上传 jpg 格式的图片。
- 出版日期需要提供日期选择及格式验证。

您现在的位置： 新知书店 > 管理员后台 > 图书管理 > 图书列表页

检索类别： 出版社　关键字： 高等教育出版社　查询　添加书籍

书名	作者	类别		
JSP程序设计（附光盘）	蒋文蓉 主编	JSP	编辑	删除
JSP编程技术与应用/网络编程系列丛书	方逵	JSP	编辑	删除
网站开发与管理	张李义，罗琳，黄晓梅　主编	网站开发	编辑	删除
XML基础与应用教程	印旻，景建萍　著	HTML XML	编辑	删除

图 8-35　“新知书店”管理员端图书信息列表页面

【任务实施】

1. 图书详细信息更新数据访问层与业务逻辑层的实现

图书详细信息更新页面需要获取图书类别和出版社名称，为防止在图书记录编辑过程中少出错，使用 DropDownList 动态绑定出版社和图书分类的全部信息供用户选择。这里的编辑图书是图书信息列表页传递过来的一本特定的图书，所以需要根据传递过来的图书 ID 号获取对应的图书详细信息。

1）图书详细信息更新数据访问层的实现

在 BookShopDAL 项目中新建 PublisherService.cs 类文件，添加一个根据 SQL 语句获取出版社名称信息列表对象的 GetPublishersBySql(string safeSql)方法，并由 GetPublishers()方法构建 SQL 语句来调用，代码如下：

```
public List<Publisher> GetPublishers()
{
    string sqlAll = "SELECT * FROM Publishers";
    return GetPublishersBySql(sqlAll);
}

private List<Publisher> GetPublishersBySql(string safeSql)
{
    List<Publisher> list = new List<Publisher>();

    DataSet ds = SqlHelper.ExecuteDataset(this.connection, CommandType.Text, safeSql);
    if (ds.Tables.Count > 0)
    {
        DataTable dt = ds.Tables[0];
        foreach (DataRow row in dt.Rows)
        {
            Publisher publisher = new Publisher();
            publisher.Id = (int)row["Id"];
            publisher.Name = (string)row["Name"];
            list.Add(publisher);
        }
    }
    return list;
}
```

在 BookShopDAL 项目的 BookService.cs 类中，添加一个根据图书 ID 获取图书信息的 GetBookById(int id)方法，代码如下：

```
public Book GetBookById(int id)
{
    string sql = "SELECT * FROM Books WHERE Id = @Id";
    int publisherId;
    int categoryId;
    Book book = null;
    using (SqlDataReader reader = SqlHelper.ExecuteReader(this.connection, CommandType.
Text,sql, new SqlParameter("@Id", id)))
    {
        if (reader.Read())
        {
            book = new Book();
            book.Id = (int)reader["Id"];
            book.Title = (string)reader["Title"];
            book.Author = (string)reader["Author"];
            book.PublishDate = (DateTime)reader["PublishDate"];
            book.ISBN = (string)reader["ISBN"];
            book.UnitPrice = (decimal)reader["UnitPrice"];
```

```
                book.ContentDescription = (string)reader["ContentDescription"];
                book.TOC = (string)reader["TOC"];
                book.Clicks = (int)reader["Clicks"];
                publisherId = (int)reader["PublisherId"];   //FK
                categoryId = (int)reader["CategoryId"];     //FK
                reader.Close();
                book.Publisher = new PublisherService().GetPublisherById(publisherId);
                book.Category = new CategoryService().GetCategoryById(categoryId);
            }
        }
        return book;
    }
```

获取图书类别信息的数据访问层在前面已经介绍过，此处不再赘述。

2）图书详细信息更新业务逻辑层的实现

在 BookShopBLL 项目中的 BookManager.cs 类文件中，添加如下代码：

```
public Book GetBookById(int id)
{
    return new BookService().GetBookById(id);
}
```

在 BookShopBLL 项目中的 PublisherManager.cs 类文件中，添加如下代码：

```
public List<Publisher> GetPublishers()
{
    return new PublisherService().GetPublishers();
}
```

2. 图书详细信息更新表示层的实现

1）表示层页面设计

在任务 8-2-1 的基础上为图书列表页 BookList.aspx 中的 GridView 控件对象 gvBooks 添加操作列，这里通过设置 HyperLinkField 列类型字段实现页面跳转，如图 8-36 所示。

这里设置 DataNavigateUrlFields 属性值为图书 ID，表示目标页面传递的参数来自 ID 字段，将 DataNavigateUrlFormatString 字段设置为"BookEdit.aspx? id={0}"，表示要链接的目标页面页面为同一目录下的 BookEdit.aspx 页面，参数为 ID，参数即为刚才设置的 DataNavigateUrlFields 字段的的值，设置后，打开源视图可以看到增加了以下代码：

```
<asp:HyperLinkField DataNavigateUrlFormatString = "BookEdit.aspx?id = {0}" Text = "编辑"
      DataNavigateUrlFields = "id" HeaderText = "操作" />
```

在 Web 站点项目的文件夹 Admin 下，根据后台母版页 Admin.master 新建图书详细信息更新内容页面 BookEdit.aspx，根据如图 8-37 所示的设计要求，从工具箱拖入 FileUpload、DropDownList、CKeditor 等控件至页面。

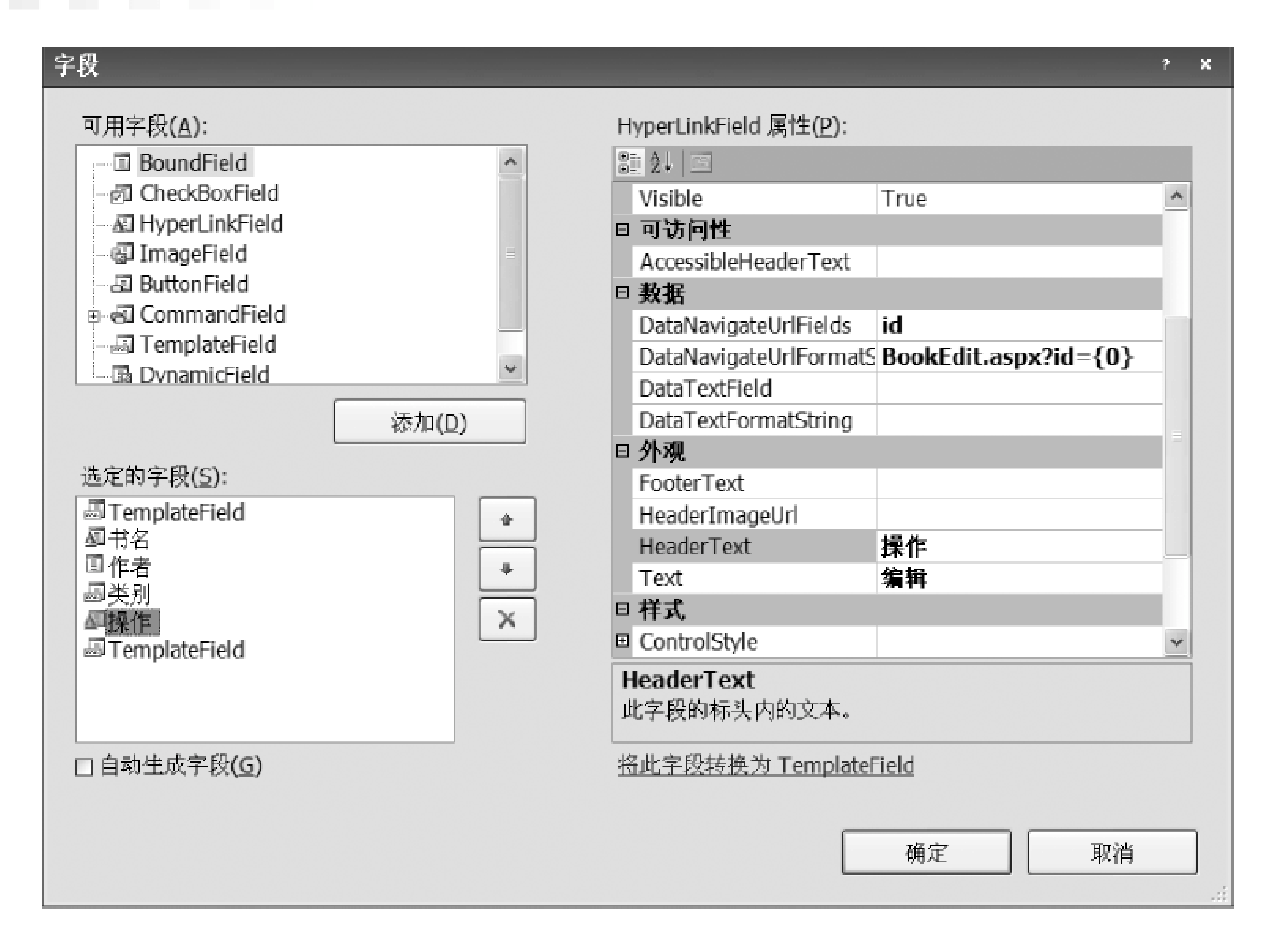


图 8-36　设置 HyperLinkField 属性

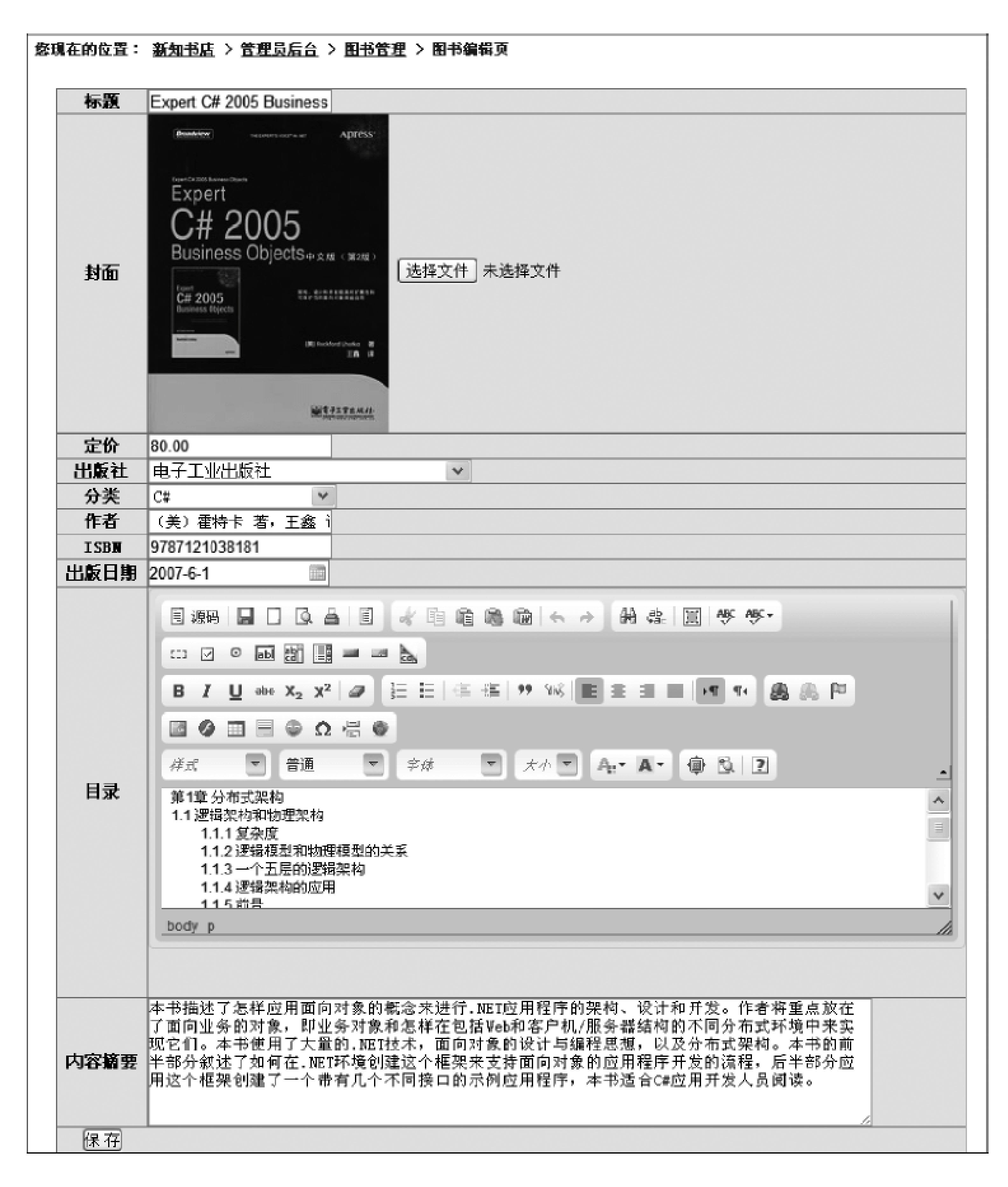


图 8-37　“新知书店”图书详细信息页面

BookEdit.aspx 页面代码如下：

```
<asp:Content ID="Content1" ContentPlaceHolderID="cphAdmin" runat="Server">
  <script language="javascript" type="text/javascript" src="../My97DatePicker/
WdatePicker.js"
    charset="gb2312">
  </script>
  <table cellspacing="1" cellpadding="3" class="table_edit">
    <tr>
        <th>标题</th>
        <td>
            <asp:TextBox ID="txtTitle" runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator ID="rfvTitle" runat="server" ControlToValidate
="txtTitle" ErrorMessage="标题不可为空!"></asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <th>封面</th>
        <td>
            <asp:Image ID="imgBook" runat="server" />
            <asp:FileUpload ID="fulBook" runat="server" />
            <asp:RequiredFieldValidator ID="rfvBookImage" runat="server" ControlToValidate=
            "fulBook" ErrorMessage="封面不可为空!"></asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <th>定价</th>
        <td>
            <asp:TextBox ID="txtPrice" runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator ID="rfvprice" runat="server" ControlToValidate
="txtPrice" ErrorMessage="定价不可为空!"></asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <th>出版社</th>
        <td>
            <asp:DropDownList ID="ddlPublisher" runat="server" DataTextField="Name"
                  DataValueField="Id"></asp:DropDownList>
        </td>
    </tr>
    <tr>
        <th>分类</th>
        <td>
            <asp:DropDownList ID="ddlCategory" runat="server" DataTextField="Name"
                  DataValueField="Id"></asp:DropDownList>
```

```
            </td>
        </tr>
        <tr>
            <th>作者</th>
            <td>
                <asp:TextBox ID = "txtAuthor" runat = "server"></asp:TextBox>
                <asp:RequiredFieldValidator ID = "rfvAuthor" runat = "server"
                  ControlToValidate =
                "txtAuthor"  ErrorMessage = "作者名不可为空!"></asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <th> ISBN </th>
            <td>
                <asp:TextBox ID = "txtISBN" runat = "server"></asp:TextBox>
                <asp:RequiredFieldValidator ID = "rfvIsbn" runat = "server" ControlToValidate =
                    "txtISBN"ErrorMessage = "ISBN 不可为空!"></asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <th>出版日期</th>
            <td>
                <asp:TextBox ID = "txtPublishDate" runat = "server" CssClass = "Wdate"
                      onfocus = "WdatePicker()"></asp:TextBox>
                <asp:RequiredFieldValidator ID = "rfvPublishDate" runat = "server"
                            ControlToValidate = "txtPublishDate"
                            ErrorMessage = "出版日期不可为空!"></asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <th>目录</th>
            <td>
                <CKEditor:CKEditorControl ID = "ftbToc" runat = "server" Width = "680px"
                          Height = "100px"></CKEditor:CKEditorControl><br>
                 <asp:RequiredFieldValidator ID = "rfvToc" runat = "server" ControlToValidate =
"ftbToc"
                    ErrorMessage = "目录不可为空!"></asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <th>内容摘要</th>
            <td>
                <asp:TextBox ID = "txtDesc" runat = "server" Height = "105px" TextMode = "MultiLine"
                      Width = "600px"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td colspan = "2">
```

```
            <asp:Button ID="bntSave" runat="server" Text="保 存" OnClick="bntSave_Click" />
        </td>
    </tr>
  </table>
</asp:Content>
```

注意：除了 HyperLinkField 之外，还可以通过设置 CommandFile、TemplateFiled 的方法进行页面间的跳转，GridView 功能强大，在实际工作中，常常根据用户需求选择最合适的做法。

2）编程实现图书详细信息更新操作

图书详细信息页面中，使用 Request.QueryString["id"]获取图书列表页传递过来的参数 Id，然后根据图书 Id 检索图书详细信息并绑定到图书详细信息页面 BookEdit.aspx 相应的控件上，图书详细信息页的后置代码文件 BookEdit.aspx.cs 中编程实现图书详细信息更新，代码如下：

```
/// <summary>
/// 页面加载事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        ddlCategory.DataSource = new CategoryManager().GetCategories();
        ddlCategory.DataBind();
        ddlPublisher.DataSource = new PublisherManager().GetPublishers();
        ddlPublisher.DataBind();
        if (Request.QueryString["id"] == null)
        {
            this.imgBook.Visible = false;
        }
        else
        {
            this.rfvBookImage.Visible = false;
        }
        this.BindData();
    }
}

/// <summary>
/// 单击"保存"按钮事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void bntSave_Click(object sender, EventArgs e)
```

```
{
    string FileName = this.fulBook.FileName;
    if (FileName.Trim().Trim().Length != 0)
    {
        string strpath = Server.MapPath("~/images/BookCovers/" + txtISBN.Text.Trim() + ".jpg");
        fulBook.PostedFile.SaveAs(strpath);      //把图片保存在此路径中
    }
    BookManager manger = new BookManager();
    Book book = new Book();
    book.Author = txtAuthor.Text;
    //Category 为图书分类实体类,在 Category.cs 中有构造方法 public Category(int id,string
name)
     book.Category = new Category(int.Parse(ddlCategory.SelectedValue), ddlCategory.
SelectedItem.Text);
    book.ContentDescription = txtDesc.Text;
    book.ISBN = txtISBN.Text;
    book.PublishDate = DateTime.Parse(txtPublishDate.Text);
    //Publisher 为出版社信息实体类,在 Publisher.cs 中有构造方法 public Publisher(int id,
string name)
     book.Publisher = new Publisher(int.Parse(ddlPublisher.SelectedValue),ddlPublisher.
SelectedItem.Text);
    book.Title = txtTitle.Text;
    book.TOC = ftbToc.Text;
    book.UnitPrice = decimal.Parse(txtPrice.Text);
    if (Request.QueryString["id"] != null)
    {
        book.Id = Convert.ToInt32(Request.QueryString["id"]);
        if (manger.ModifyBook(book))
            Page.RegisterClientScriptBlock("", "<script>alert('书籍修改成功!')</script>");
    }
    else
    {
        manger.AddBook(book);
        Response.Redirect("~/admin/BookList.aspx");
        //Page.RegisterClientScriptBlock("", "<script>alert('书籍添加成功!')</script>");
    }
}

/// <summary>
/// 根据图书 Id 绑定图书信息到页面对应控件
/// </summary>
private void BindData()
{
    if (Request.QueryString["id"] != null)
    {
        Book book = new BookManager().GetBookById(Convert.ToInt32(Request.QueryString["id"]));
        this.txtAuthor.Text = book.Author;
        this.txtTitle.Text = book.Title;
        this.ddlPublisher.SelectedValue = book.Publisher.Id.ToString();
        this.ddlCategory.SelectedValue = book.Category.Id.ToString();
```

```
            this.txtISBN.Text = book.ISBN;
            this.txtPublishDate.Text = book.PublishDate.ToShortDateString();
            this.txtPrice.Text = string.Format("{0:f2}", book.UnitPrice);
            this.txtDesc.Text = book.ContentDescription;
            this.ftbToc.Text = book.TOC;
            this.imgBook.ImageUrl = "~/Images/BookCovers/" + book.ISBN + ".jpg";
            //this.imgBook.ImageUrl = StringUtility.CoverUrl(book.ISBN);    //效果与上一行代
                                                                            //码一样
        }
    }
```

任务 8-2-3　实现“新知书店”管理员端的图书添加功能

【任务描述】

实现图 8-35 中的添加书籍功能。

【任务实施】

1. 思路分析

“新知书店”的管理端除可以修改图书信息外，还需要图书的添加功能，可以在图书信息列表页面添加一个“添加书籍”的链接，当管理员单击该链接时，页面将跳转到新增图书信息页面。新增图书页面与图书信息修改页面相比，除了不需要传递图书编号这个参数和不需要检索当前图书信息外，其他没有区别，可以使用图书信息修改页面完成图书添加功能，这里为了区分，把新增图书信息页面取名为 BookAdd.aspx（代码相同）。可以通过 Request.QueryString[“Id”]是否为空来判断当前执行的是修改操作还是新增操作。如果执行的是新增操作，就将图书信息插入到 Books 表中。

2. 图书信息添加数据访问层的实现

在 BookShopDAL 项目的 BookService.cs 类中，添加一个添加图书详细信息的方法 AddBook(Book book)，代码如下：

```
public void AddBook(Book book)
{
    //拼接 SQL 语句
    string sql = "INSERT Books (Title, Author, PublisherId, PublishDate, ISBN, UnitPrice,
ContentDescription, TOC, CategoryId)" + "VALUES (@Title, @Author, @PublisherId, @PublishDate,
@ISBN, @UnitPrice, @ContentDescription, @TOC, @CategoryId)";
    sql += " ; SELECT @@IDENTITY";   //获取最新添加的用户 Id
    SqlParameter[] para = new SqlParameter[]
        {
            new SqlParameter("@PublisherId", book.Publisher.Id), //FK
            new SqlParameter("@CategoryId", book.Category.Id),   //FK
            new SqlParameter("@Title", book.Title),
            new SqlParameter("@Author", book.Author),
            new SqlParameter("@PublishDate", book.PublishDate),
```

```
                new SqlParameter("@ISBN", book.ISBN),
                new SqlParameter("@UnitPrice", book.UnitPrice),
                new SqlParameter("@ContentDescription", book.ContentDescription),
                new SqlParameter("@TOC", book.TOC),
            };
        book.Id = Convert.ToInt32(SqlHelper.ExecuteScalar(this.connection, CommandType.Text,
    sql, para));
    }
```

代码与任务 6-2-3 实现用户注册添加用户信息很相似。

3. 图书信息添加表示层的实现

添加图书信息页面 BookAdd.aspx 与修改图书详细信息的页面 BookEdit.aspx 一样，不需要修改，后置代码文件亦不需要修改。单击页面“保存”按钮的事件方法部分代码如下：

```
if (Request.QueryString["id"] != null)
{
    book.Id = Convert.ToInt32(Request.QueryString["id"]);
    if (manger.ModifyBook(book))
   Page.RegisterClientScriptBlock("", "<script>alert('书籍修改成功!')</script>");
}
else
{
    manger.AddBook(book);
    Response.Redirect("~/admin/BookList.aspx");
    //Page.RegisterClientScriptBlock("", "<script>alert('书籍添加成功!')</script>");
}
```

任务 8-2-4　实现“新知书店”前台图书列表显示功能

【任务描述】

- 为“新知书店”添加图书列表显示功能，要求显示点击率前五位的书籍信息，包括图书封面、图书标题、作者、内容描述、价格信息。
- 图书标题和封面要求实现链接功能，内容描述要求只显示前 180 个字符。
- 如果当前图书信息为空，页面将跳转到前台默认页面，效果如图 8-38 所示。

【任务实施】

1. 图书列表数据访问层与业务逻辑层的实现

DataList 控件能够实现灵活复杂的页面布局，可以通过它来做图书列表，要完成这一功能，需先从数据访问层和业务逻辑层着手。本任务需要取出的图书信息类似任务 8-2-1 中管理端图书查询所需要的图书信息。

（1）图书列表数据访问层的实现。

在 BookShopDAL 项目的 BookService.cs 类中，添加一个获取 5 本点击率最高的图书



图 8-38 “新知书店”前台图书列表显示页面

信息列表对象的方法 GetClickMoreBooks(int count)，代码如下：

```
public List < Book > GetClickMoreBooks(int count)
{
    string sqlAll = "SELECT TOP " + count + " * FROM Books ORDER BY Clicks DESC";
    return GetBooksBySql(sqlAll);
}
public List < Book > GetBooksBySql(string safeSql)
{
     //代码在示例 8 - 2 的表 8 - 4 中,此处省略
}
```

(2) 在 BookShopBLL 项目的类文件 BookManager.cs 中，添加代码如下：

```
public List < Book > GetClickMoreBooks(int count)
{
    return new BookService().GetClickMoreBooks(count);
}
```

2. 图书查询表示层的实现

1) 表示层页面设计

在站点项目 Web 下根据前台母版页 Common.master 创建内容页 preBookList.aspx（注意：BookList.aspx 是下一个任务的页面），并从工具箱拖入 DataList 控件至页面并定义 ItemTemplate 模板，代码如下：

```
<asp:Content ID = "Content2" ContentPlaceHolderID = "cphHeader" runat = "Server">
<link href = "Css/channel.css" rel = "stylesheet" type = "text/css" />
</asp:Content>
<asp:Content ID = "Content1" ContentPlaceHolderID = "cphContent" runat = "Server">
  您现在的位置:
  <div class = "main">
    <asp:DataList ID = "dlBooks" runat = "server">
      <ItemTemplate>
        <table id = "tbBooks" class = "list_area">
            <tr>
                <td rowspan = "2" class = "td_left">
                    <a href = "BookDetail.aspx?bid = <% # Eval("Id") %>">
                        <% --<img alt = "" src = "<% # GetUrl(Eval("ISBN").ToString())
 %>" /> -- %>
                        <img alt = "" src = '<% # Eval("ISBN","Images/BookCovers/{0}.
jpg") %>'
                    </a>
                </td>
                <td class = "td_right">
                    <a href = "BookDetail.aspx?bid = <% # Eval("Id") %>" name = "link_prd_
name"
                          target = "_blank" class = "b_title" id = "link_prd_name">
                          <% # Eval("Title") %></a>
                </td>
            </tr>
            <tr>
                <td align = "left">
                    <span><% # Eval("Author") %></span><br /><br /></span>
                    <span><% # StringUtility.CutString(Eval("ContentDescription"), 180) %>
</span>
                </td>
            </tr>
            <tr>
                <td align = "right" colspan = "2">价格:
                    <span class = "red"><% # StringUtility.ToMoney(Eval("UnitPrice")) %>
</span>
                </td>
            </tr>
        </table>
      </ItemTemplate>
      <SeparatorTemplate></SeparatorTemplate>
    </asp:DataList>
  </div>
</asp:Content>
```

获取图书封面用<img alt="" src='<%#Eval("ISBN","Images/BookCovers/{0}.jpg")%>'的方式,它是Eval()的一个重载方法,非常重要,要使用熟练。Eval("ContentDescription")获取图书内容描述,将它作为CutString(object content, int num)方法的参数,用于截取前180个字符。CutString方法定义在Web站点项目的文件夹App_Code下的StringUtility.cs类

文件中，代码如下：

```
/// <summary>
/// 截断字符串
/// </summary>
public static string CutString(object content, int num)
{
    if (content.ToString().Length > num - 2)
        return content.ToString().Substring(0, num - 2) + "...";
    else
        return content.ToString();
}
```

2）编程实现图书列表相关功能

在图书列表页的后置代码文件 preBookList.aspx.cs 中编程实现图书列表功能，添加代码如下：

```
⋮
using BookShop.BLL;
using BookShop.Models;
public partial class preBookList : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            this.DataBind();
        }
    }

    private void DataBind()
    {
        BookManager manager = new BookManager();
        List<Book> list = manager.GetClickMoreBooks(5);
        this.dlBooks.DataSource = list;
        this.dlBooks.DataBind();
    }

    /// <summary>
    /// 获得封面的 url
    /// </summary>
    public string GetUrl(string isbn)
    {
        return "Images/BookCovers/" + isbn.ToString() + ".jpg";
    }
}
```

任务 8-2-5 实现“新知书店”前台图书列表显示的排序和分页

【任务描述】

• 实现每页显示五条图书记录，分页功能如图 8-39 所示。

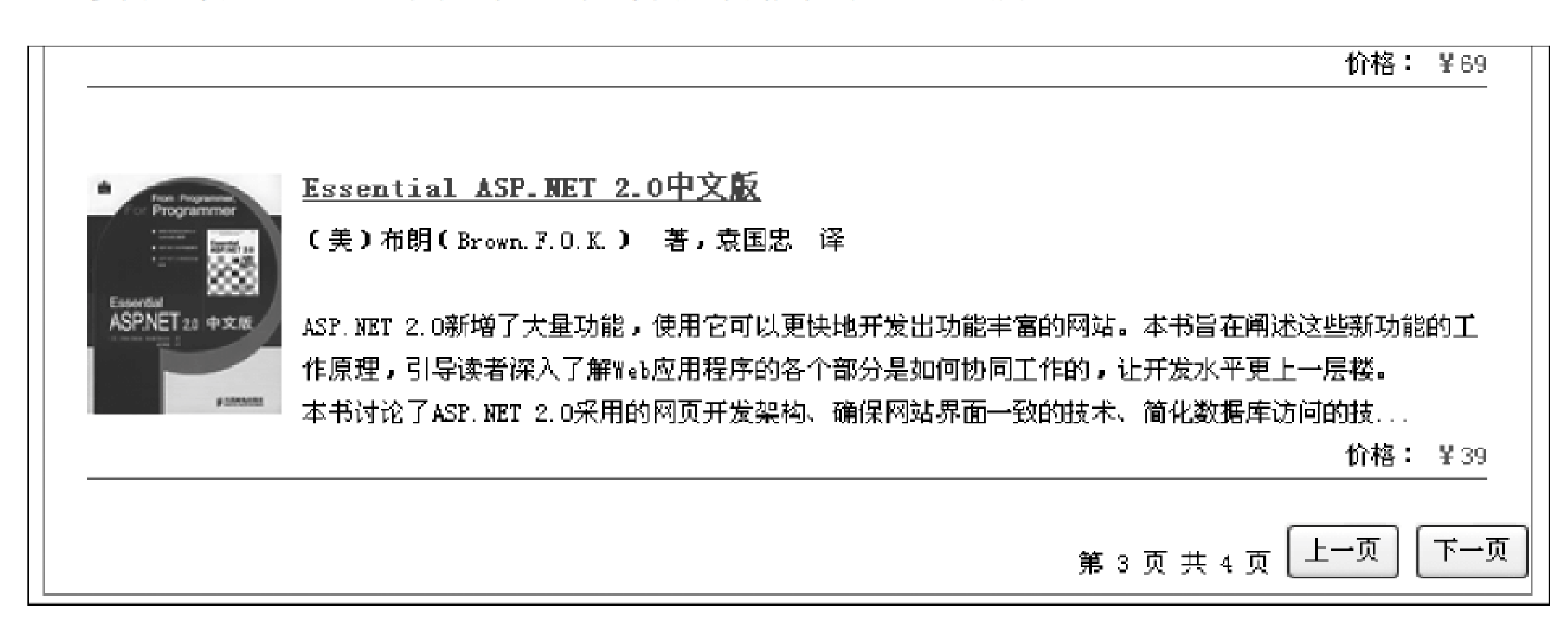


图 8-39 “新知书店”前台图书列表分页页面

• 实现“新知书店”的图书列表页按“按出版日期排序”和“按价格排序”功能如图 8-40 所示。



图 8-40 “新知书店”前台图书列表排序功能

• 实现单击 TreeView 的分类名，显示该分类下的全部图书信息。

【任务实施】

1. 图书列表排序和分页数据访问层与业务逻辑层的实现

DataList 控件没有提供内置分页功能，有时候不方便。目前有很多添加分页的方法，比如使用存储过程来控制每页的数据读取，这种方式效率高但比较复杂，本任务采用 PagedDataSource 对象给 DataList 实现分页。

1）图书列表排序和分页数据访问层的实现

在 BookShopDAL 项目的 BookService.cs 类中，添加一个根据图书分类 ID 号和排序方式获取图书信息列表对象的方法 GetBooks(int categoryId, string sortField)，代码如下：

```
/// <summary>
/// 根据分类号和排序字段获取图书信息
/// </summary>
/// <param name="categoryId"></param>
/// <param name="sortField"></param>
/// <returns></returns>
public List<Book> GetBooks(int categoryId, string sortField)
{
    List<Book> list = new List<Book>();
    SqlParameter[] paras = new SqlParameter[]
    {
        new SqlParameter("@CategoryId", categoryId),
        new SqlParameter("@SortField", sortField)
    };
    DataSet ds = SqlHelper.ExecuteDataset(this.connection, CommandType.StoredProcedure, "sp
_GetBooksByCategoryIdAndSortField", paras);
    if (ds.Tables.Count > 0)
    {
        DataTable dt = ds.Tables[0];
        foreach (DataRow row in dt.Rows)
        {
            Book book = new Book();
            book.Id = (int)row["Id"];
            book.Title = (string)row["Title"];
            book.Author = (string)row["Author"];

            if (dt.Columns.Contains("UnitPrice"))
                book.UnitPrice = (decimal)row["UnitPrice"];
            if (dt.Columns.Contains("ContentDescription"))
                book.ContentDescription = (string)row["ContentDescription"];
            if (dt.Columns.Contains("ISBN"))
                book.ISBN = (string)row["ISBN"];
            if (dt.Columns.Contains("Clicks"))
                book.Clicks = (int)row["Clicks"];
            if (dt.Columns.Contains("PublishDate"))
                book.PublishDate = (DateTime)row["PublishDate"];
            if (dt.Columns.Contains("CategoryId"))
                book.Category = new CategoryService().GetCategoryById((int)row["CategoryId"]);
                book.Publisher = new PublisherService().GetPublisherById((int)row
["PublisherId"]);        //FK
            list.Add(book);
        }
    }
    return list;
}
```

其中 sp_GetBooksByCategoryIdAndSortField 为根据根据图书分类 Id 号和排序方式获取图书信息列表的存储过程，代码如下：

```
ALTER PROCEDURE [dbo].[sp_GetBooksByCategoryIdAndSortField]
@SortField VARCHAR(20),      --排序方式
@CategoryId INT              --图书类别
AS
--根据分类号和排序关键字获取图书列表
IF(@SortField = 'PublishDate')
BEGIN
    select * from books where CategoryId = @CategoryId ORDER BY PublishDate
END
ELSE IF(@SortField = 'UnitPrice')
BEGIN
    select * from books where CategoryId = @CategoryId ORDER BY UnitPrice
END
```

2）图书列表排序和分页业务逻辑层的实现

在 BookShopBLL 项目的类文件 BookManager.cs 中，添加代码如下：

```
public List<Book> GetBooks(int category, string sortField)
{
    return new BookService().GetBooks(category, sortField);
}
```

2. 图书查询表示层的实现

1）表示层页面设计

在站点项目 Web 下根据前台母版页 Common.master 创建内容页 BookList.aspx，并从工具箱拖入 DataList 控件至页面并定义 ItemTemplate 模板，ItemTemplate 模板代码与任务 8-2-4 中的页面 preBookList.aspx 定义的一致，拖入 DropDownList 控件和 Label 控件，代码如下：

```
<asp:Content ID = "Content1" ContentPlaceHolderID = "cphContent" runat = "Server">
<div class = "main">
    <div class = "list_asc">
        <! -- choice order type -- >
        <div class = "type_choice f_left">
            排序方式
            <asp:DropDownList ID = "ddlOrder" runat = "server" AutoPostBack = "true"
OnSelectedIndexChanged = "ddlOrder_SelectedIndexChanged">
                <asp:ListItem Value = "1">按出版日期排序</asp:ListItem>
                <asp:ListItem Value = "2">按价格排序</asp:ListItem>
            </asp:DropDownList>
        </div>
    </div>
    <asp:DataList ID = "dlBooks" runat = "server">
                <% -- ItemTemplate 模板代码在任务 8-2-4 中定义过,此处省略 -- %>
```

```
    </asp:DataList>
    <div class="pages">
        <asp:Label runat="server" ID="lblCurrentPage"></asp:Label>
        <asp:Button ID="btnPrev" runat="server" Text="上一页" OnClick="btnPrev_Click" />
        <asp:Button ID="btnNext" runat="server" Text="下一页" OnClick="btnNext_Click" />
    </div>
</div>
</asp:Content>
```

2）编程实现图书列表排序和分页相关功能

在图书列表页排序和分页的后置代码文件 BookList.aspx.cs 中编程实现图书列表排序和分页的功能，代码如下：

```
 ⋮
using BookShop.Models;
using BookShop.BLL;
public partial class BookList : System.Web.UI.Page
{
    private int PageSize = 5;
    /// <summary>
    /// 当前页
    /// </summary>
    public int CurrentPageIndex
    {
        set
        {
            ViewState["CurrentPageIndex"] = value;
        }
        get
        {
            return Convert.ToInt32(ViewState["CurrentPageIndex"]);
        }
    }
    /// <summary>
    /// 总页数
    /// </summary>
    public int PageCount
    {
        set
        {
            ViewState["PageCount"] = value;
        }
        get
        {
            return Convert.ToInt32(ViewState["PageCount"]);
        }
    }
    /// <summary>
```

```
/// 当前分类 Id
/// </summary>
private int CategoryId
{
    get
    {
        return (int)ViewState["CategoryId"];
    }
    set
    {
        ViewState["CategoryId"] = value;
    }
}
/// <summary>
/// 当前排序字段
/// </summary>
private string Order
{
    get
    {
        if (ViewState["Order"] == null)
            return "PublishDate";
        return (string)ViewState["Order"];
    }
    set
    {
        ViewState["Order"] = value;
    }
}
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)                        //首次加载,赋初值
    {
        this.CurrentPageIndex = 1;
        try
        {
            this.CategoryId = Convert.ToInt32(Request.QueryString["typeid"]);
        }
        catch
        {
            this.CategoryId = -1;
        }
        this.BindList();
    }
}
public void BindList()                      //绑定图书信息的方法
{
    BookManager manager = new BookManager();
    List<Book> list = manager.GetBooks(CategoryId, Order);
    PagedDataSource pds = new PagedDataSource(); //定义一个 PagedDataSource 实例
```

```
        pds.AllowPaging = true;                     //设置数据绑定控件启用分页
        pds.CurrentPageIndex = CurrentPageIndex - 1;  //使用状态保持保存当前页数
        if (list.Count == 0)
            Response.Redirect("~/Default.aspx");
        pds.DataSource = list;                      //指定 PagedDataSource 数据源
        pds.PageSize = this.PageSize;               //设置每页记录数
        this.PageCount = pds.PageCount;
        this.lblCurrentPage.Text = "第" + CurrentPageIndex + "页,共" + this.PageCount + "页";
        this.dlBooks.DataSource = pds;     //将 DataList 的数据源设置成 PagedDataSource
        this.dlBooks.DataBind();
    }
    /// <summary>
    /// 下拉列表选项发生改变的事件方法(排序方式变化)
    /// </summary>
    protected void ddlOrder_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (ddlOrder.SelectedValue == "1")
        {
            this.Order = "PublishDate";
        }
        else
        {
            this.Order = "UnitPrice";
        }
        this.CurrentPageIndex = 1;
        this.BindList();
    }
    /// <summary>
    /// "翻页"标签的状态
    /// </summary>
    private void SetEnable(int pageCount)
    {
        this.btnPrev.Enabled = true;
        this.btnNext.Enabled = true;
        if (this.CurrentPageIndex == 1)
            btnPrev.Enabled = false;
        if (this.CurrentPageIndex == pageCount)
            btnNext.Enabled = false;
    }
    /// <summary>
    /// "下一页"按钮事件方法
    /// </summary>
    protected void btnNext_Click(object sender, EventArgs e)
    {
        this.CurrentPageIndex++;
        this.BindList();
    }
    /// <summary>
    /// "上一页"按钮事件方法
    /// </summary>
```

```
        protected void btnPrev_Click(object sender, EventArgs e)
        {
            this.CurrentPageIndex--;
            this.BindList();
        }
        /// <summary>
        /// 获得封面的 url
        /// </summary>
        public string GetUrl(string isbn)
        {
            return "Images/BookCovers/" + isbn.ToString() + ".jpg";
        }
    }
```

任务 8-2-6　搭建“新知书店”前台图书详细信息显示页面

【任务描述】

实现单击图 8-38 中的图书的封面或标题时，链接到页面 BookDetailsView.aspx 显示该图书的详细信息，效果如图 8-45 所示。

【任务实施】

1. 图书详细信息显示数据访问层与业务逻辑层的实现

DetailsView 控件一次可以显示一条记录，当需要详细显示数据库文件中的某一条记录时，使用 DetailsView 控件就非常方便，当然，数据访问层和业务逻辑层要能实现根据图书 Id 值从图书表 Books 里查找相应的图书记录，并返回一个图书对象。

1）图书详细信息显示数据访问层的实现

在 BookShopDAL 项目的 BookService.cs 类中，添加一个根据图书 Id 值获取一本图书信息对象的方法 GetBookById(int id)，代码如下：

```
public Book GetBookById(int id)
{
    string sql = "SELECT * FROM Books WHERE Id = @Id";
    int publisherId;
    int categoryId;
    Book book = null;
    using (SqlDataReader reader = SqlHelper.ExecuteReader(this.connection, CommandType.
Text, sql, new SqlParameter("@Id", id)))
    {
        if (reader.Read())
        {
            book = new Book();
            book.Id = (int)reader["Id"];
            book.Title = (string)reader["Title"];
            book.Author = (string)reader["Author"];
            book.PublishDate = (DateTime)reader["PublishDate"];
            book.ISBN = (string)reader["ISBN"];
```

```
                book.UnitPrice = (decimal)reader["UnitPrice"];
                book.ContentDescription = (string)reader["ContentDescription"];
                book.TOC = (string)reader["TOC"];
                book.Clicks = (int)reader["Clicks"];
                publisherId = (int)reader["PublisherId"];     //FK
                categoryId = (int)reader["CategoryId"];       //FK
                reader.Close();
                book.Publisher = new PublisherService().GetPublisherById(publisherId);
                book.Category = new CategoryService().GetCategoryById(categoryId);
            }
        }
        return book;
    }
```

2）图书详细信息显示业务逻辑层的实现

在 BookShopBLL 项目的类文件 BookManager.cs 中，添加代码如下：

```
public Book GetBookById(int id)
{
    return new BookService().GetBookById(id);
}
```

2. 图书详细信息显示表示层的设计与实现

（1）基于前台母版页 Common.master 创建内容页 BookDetailsView.aspx，并从工具箱拖入 DetailsView 控件至页面。选中 DetailsView 控件，点击右上角的箭头符号，弹出“DetailsView 任务”窗口，在“DetailsView 任务”窗口的“选择数据源”下拉列表框中选“新建数据源”选项，弹出“数据源配置向导”对话框，在对话框的“应用程序从哪里获取数据(W)?”选项组中选择“对象”选项，为数据源指定 ID 为 odsBook，单击“下一步”按钮，进入“配置数据源”对话框，在“选择业务对象”下拉列表框中选择 BookShop.BLL.BookManager 选项，如图 8-41 所示。



图 8-41　ObjectDataSource 控件的“配置数据源”对话框

(2) 单击“下一步”按钮，在“定义数据方法”窗口的 SELECT 选项卡的“选择方法”下拉列表框中选择“GetBookById(Int32 id)，返回 Book”选项，如图 8-42 所示，如果需要使用 DetailsView 空间进行图书信息的修改、删除和添加，可依次为“定义数据方法”的 UPDATE、INSERT、DELETE 选项卡选择方法。

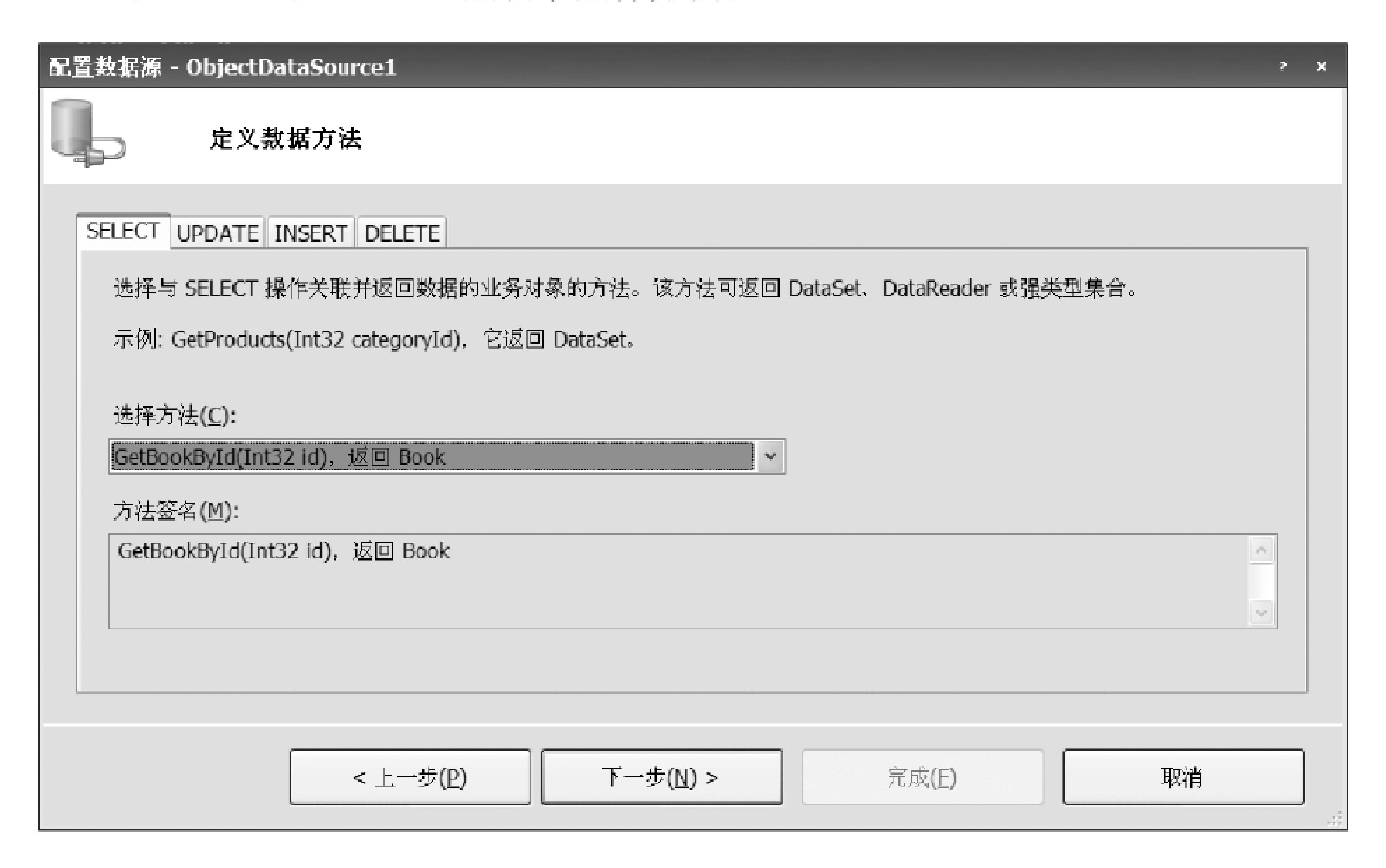


图 8-42 ObjectDataSource 控件的“定义数据方法”窗口

(3) 单击“下一步”按钮，在“定义参数”窗口中将“参数源”选择为 QueryString，将 QueryStringField 设置为 bid(bid 为 BookList.aspx 页面传过来的参数)，如图 8-43 所示。



图 8-43 ObjectDataSource 控件的“定义参数”窗口

单击“完成”按钮，完成 DetailsView 控件 ObjectDataSource 数据源的配置。代码如下：

```
<asp:ObjectDataSource ID = "odsBook" runat = "server" SelectMethod = "GetBookById"
    TypeName = "BookShop.BLL.BookManager">
    <SelectParameters>
        <asp:QueryStringParameter DefaultValue = "6088" Name = "id" QueryStringField = "bid"
            Type = "Int32" />
    </SelectParameters>
</asp:ObjectDataSource>
```

(4) 至此，在图书列表页 BookList.aspx 中单击某一图书的标题，链接到选中图书的详细信息显示页面 BookDetailsView.aspx。

从显示页面可以看出，图书的详细信息内容均已经显示出来，但比较混乱，为达到显示美观的效果，对 DetailsView 控件的相关属性进行设置，并在“DetailsView 任务”窗口中选择“编辑字段”选项，在“字段”对话框中对自动以数据表字段自动生成的对应字段进行调整，如图 8-44 所示，具体设置方式与 GridView 的字段设置大同小异，在此不予赘述。需要注意的是，为了显示“购买”按钮，把“单价”字段转换成了模板。设置完成后重新运行，效果如图 8-45 所示。

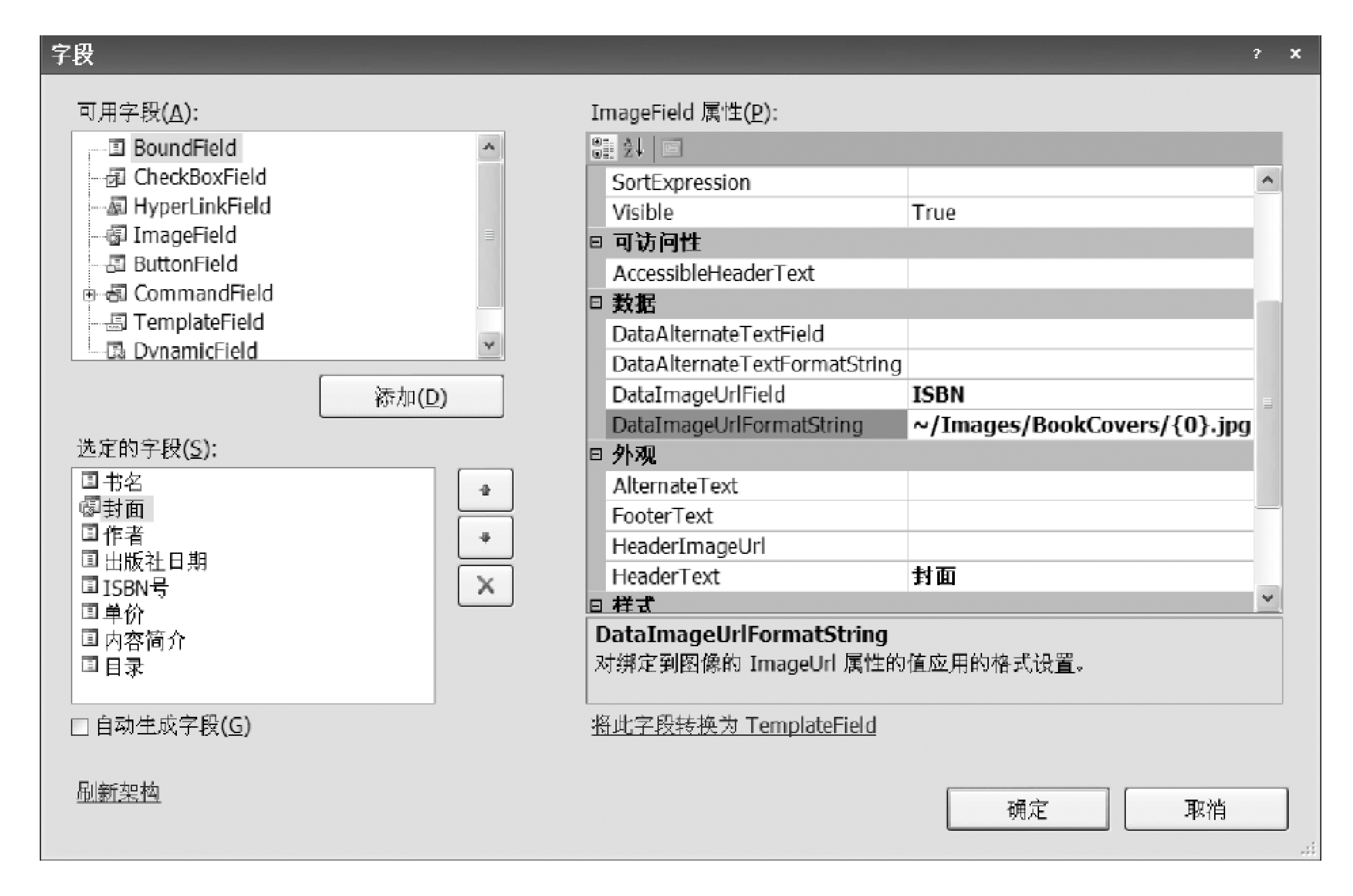


图 8-44　DetailsView 控件的“字段”对话框

切换到 BookDetailsView.aspx 页面源视图，可以看到设置完后生成了如下代码：

```
<asp:DetailsView ID = "dvShowBook" runat = "server" AutoGenerateRows = "False" CellPadding = "4"
    DataSourceID = "odsBook" ForeColor = "#333333" GridLines = "None" Height = "50px"
    HorizontalAlign = "Center"  Width = "650px">
    <AlternatingRowStyle BackColor = "White" ForeColor = "#284775" />
```



图 8-45 “新知书店”前台图书详细信息页面运行效果

```
<CommandRowStyle BackColor = "#E2DED6" Font - Bold = "True" />
<EditRowStyle BackColor = "#999999" />
<FieldHeaderStyle BackColor = "#E9ECF1" Font - Bold = "True" HorizontalAlign = "Center"
    VerticalAlign = "Middle"  Width = "80px" Wrap = "False" />
<Fields>
    <asp:BoundField DataField = "Title" HeaderText = "书名" SortExpression = "Title" />
    <asp:ImageField DataImageUrlField = "ISBN" DataImageUrlFormatString =
        "~/Images/BookCovers/{0}.jpg"  HeaderText = "封面">
    </asp:ImageField>
    <asp:BoundField DataField = "Author" HeaderText = "作者" SortExpression = "Author" />
    <asp:BoundField DataField = "PublishDate" HeaderText = "出版社日期"
                SortExpression = "PublishDate" />
    <asp:BoundField DataField = "ISBN" HeaderText = "ISBN 号" SortExpression = "ISBN" />
    <asp:TemplateField HeaderText = "单价" SortExpression = "UnitPrice">
      <ItemTemplate>
          <asp:Label ID = "Label1" runat = "server"
                         Text = '<%# Bind("UnitPrice", "{0:C}") %>'></asp:Label>
          <a href = '<%#Eval("id","/ShoppingCart.aspx? = {0}") %>'>
                                   <img src = "Images/Common/buy.gif" /></a>
        </ItemTemplate>
    </asp:TemplateField>
```

```
            <asp:BoundField DataField = "ContentDescription" HeaderText = "内容简介"
                    SortExpression = "ContentDescription" />
            <asp:BoundField DataField = "TOC" HeaderText = "目录" HtmlEncode = "False"
                    SortExpression = "TOC" />
        </Fields>
        <FooterStyle BackColor = "#5D7B9D" Font - Bold = "True" ForeColor = "White" />
        <HeaderStyle BackColor = "#5D7B9D" Font - Bold = "True" ForeColor = "White" />
        <PagerStyle BackColor = "#284775" ForeColor = "White" HorizontalAlign = "Center" />
        <RowStyle BackColor = "#F7F6F3" BorderStyle = "Solid" ForeColor = "#333333"
                HorizontalAlign = "Left" />
</asp:DetailsView>
```

8.3　项目实训

1. 实现“博客系统”文章管理功能

【需求说明】

- 只有注册会员才能管理文章而且只能管理自己发表的文章，实质是注册会员对自己发表的文章进行修改。
- 会员单击“文章管理”链接后，显示该会员发表的文章列表，文章列表显示文章标题及对应的“删除”和“详细”链接，效果如图 8-46 所示。
- 单击某文章的“详细”链接，显示该文章的详细信息，在文章详细信息页面单击“编辑”链接后，进入文章修改页面。

（提示：使用 GridView 控件实现）



图 8-46　“博客系统”文章列表页(ManageList. aspx)

2. 实现“博客系统”首页

【需求说明】

- 注册会员和匿名用户都可以使用此功能。
- 显示最新发表的 10 篇文章。
- 按列表的方式显示标题、文章作者及发布时间。
- 分页显示文章。
- 提供翻页功能，效果如图 8-47 所示。



图 8-47 “博客系统”首页效果(Default.aspx)

(提示：使用 Repeater 控件实现。)

8.4 单元小结

本单元阐述了几个非常重要的数据绑定控件，首先，讨论了如何使用 GridView 控件的选择功能、分页与排序、编辑与删除等操作；然后介绍通过 DataList 控件进行自定义模板绑定数据，并用 PagedDataSource 实现分页功能，本单元的知识体系如图 8-48 所示。

GridView 自身功能强大，带有丰富的数据绑定列，有许多内置事件帮助处理程序，GridView 内置了分页、排序功能，开发效率高，但是占用资源也比较高；DataList 的模板不如 GridView 多，以表的形式呈现数据，通过 DataList 可以使用不同的布局显示数据记录，本身不带分页、排序功能；Repeater 不提供任何布局，即不会生成任何 Html 代码，需要用户通过编辑模板实现布局功能，开发的周期长。在性能方面，Repeater 高于 DataList，DataList 高于 GridView。GridView 通常用于表格化数据处理，而 DataList 和 Repeater 多用于单行多列、多行单列结构的数据处理。

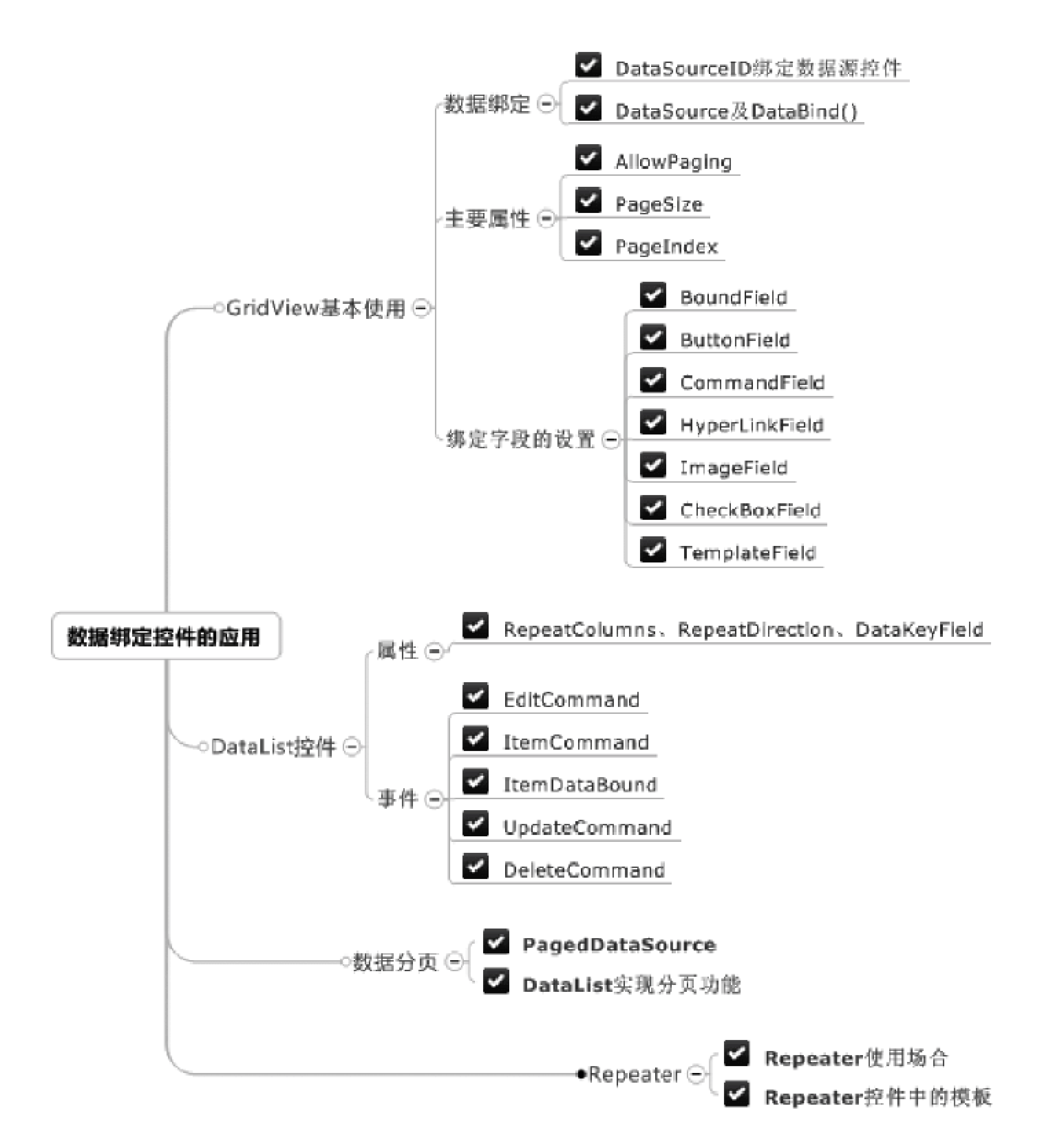


图 8-48 GridView、Repeater、DataList 控件及数据分页知识体系

8.5 单元练习题

一、选择题

1. 如果希望在 GridView 控件中显示“上一页”和“下一页”的导航栏，则 PagerSettings 的 Mode 属性为(　　)。

A. Numeric　　B. NextPrevious　　C. 上一页　　D. 下一页

2. 在 GridView 控件中，如果定制了列，有希望排序，则需要在每一列设置(　　)属性。

A. SortExpression　　B. Sort　　C. SortField　　D. DataFieldText

3. 在 ListView 控件中，如果希望每行有 4 列数据，应设置(　　)属性。

A. GroupItemCount　　B. RepeatColumn

C. RepeatLayout　　D. RepeatNumber

4. 下面关于 ListView 控件 LayoutTemplate 和 ItemTemplate 模板说法错误的是(　　)。

A. 标识定义控件的主要布局的是根模板

B. LayoutTemplate 模板包含一个占位符对象，例如表行(tr)、div 或 span 元素

C. LayoutTemplate 模板是 ListView 控件所必需的

D. LayoutTemplate 内容不必包含一个占位符控件

5. 下面关于 ListView 控件和 DataPager 控件说法错误的是(　　)。

A. ListView 就是 GridView 和 Repeater 的结合体,它既有 Repeater 控件的开放式模板,又具有 GridView 控件的编辑特性

B. ListView 控件本身不提供分页功能,但是可以通过另一个控件 DataPager 来实现分页的特性

C. 在 ListView 中,布局定义与数据绑定不可以分开在不同的模板中,只能展现数据

D. DataPager 控件能支持实现 IPageableItemContainer 接口的控件,ListView 是现有控件中唯一实现此接口的控件

6. 已知数据库连接字符串,要通过编程获取数据库中 Employees 表中数据,并绑定到 GridView 控件上。后台编写代码如下,空白处的代码应为(　　)。

```
string strcnn = ConfigurationManager.ConnectionStrings["StudentCnnString"].ConnectionString;
using (SqlConnection conn = new SqlConnection(strcnn))
{
    DataSet ds = new DataSet();
    SqlDataAdapter da = new SqlDataAdapter("select * from Employees", ________);
    da.Fill(ds);
    GridView1.______________ = ds.Tables[0];
    ______________
}
```

A. conn,DataSource,GridView1.DataBind()

B. connString,DataSource,GridView1.DataBind()

C. connString,DataSourceID,GridView1.DataBind()

D. conn,DataSourceID,GridView1.DataBind()

7. 下面的(　　)能够保持页面级的状态。

A. ViewState　　B. Session　　C. Cookie　　D. Application

8. 下面关于 DataList 控件和 Repeater 控件描述中错误的是(　　)。

A. 这两种数据控件都允许使用模板显示数据

B. Repeater 控件可以使用较少的代码实现丰富的显示效果

C. 使用 DataList 时,可以设定一些属性来进行个性化输出

D. 调用这两种控件的 DataBind()方法时实现数据与控件的绑定操作

9. 将显示在 DataList 上的数据进行分页,需要用到 PagedDataSource 类,该类封装了与分页相关的属性,其中表示总页数的属性是(　　)。

A. CurrentPageIndex　　B. Count

C. PageCount　　D. PageSize

二、填空题

1. GridView 控件的________属性表示获取或设置一个值,该值指示是否为数据源中的每个字段自动创建绑定字段。

2. ListView 控件有多种模板,其中,________标识定义控件的主要布局的根模板;________标识组布局的内容;________标识为便于区分连续项,而为交替项呈现的内容。

3. 在 GridView 控件上绑定了一列 CheckBox 控件，当表头 CheckBox 控件选中时，在 GridView 控件中的 CheckBox 全选，当取消表头 CheckBox 控件选择时，GridView 控件中的 CheckBox 控件全不选，该 GridView 控件代码如下：

```
<asp:GridView ID = "GridView1" runat = "server" AutoGenerateColumns = "False"
    DataKeyNames = "MajorId" DataSourceID = "SqlDataSource1">
    <Columns>
        <asp:TemplateField>
            <HeaderTemplate>
                <asp:CheckBox ID = "CheckBox2" runat = "server" AutoPostBack = "True" Text =
"全选" oncheckedchanged = "CheckChange" />
            </HeaderTemplate>
            <ItemTemplate>
                <asp:CheckBox ID = "CheckBox1" runat = "server" />
            </ItemTemplate>
        </asp:TemplateField>
        <asp:BoundField DataField = "MajorId" HeaderText = "MajorId" ReadOnly = "True" />
        <asp:BoundField DataField = "MajorName" HeaderText = "MajorName" />
    </Columns>
</asp:GridView>
<asp:SqlDataSource ID = "SqlDataSource1" runat = "server"
ConnectionString = "<% $ ConnectionStrings:StuConnectionString %>"
SelectCommand = "SELECT * FROM [Major]"></asp:SqlDataSource>
```

为实现题目所述的功能，必须实现 GridView 控件表头 CheckBox 控件的 oncheckedchanged 事件代码，实现代码如下。

```
protected void CheckChange(object sender, EventArgs e)
{
   CheckBox cb = (CheckBox)________;
   if (cb.Text == "全选")
    {
      foreach (GridViewRow gv in this.GridView1.Rows)
      {
            CheckBox cd = (CheckBox)gv.FindControl("________");
            cd.Checked = cb.Checked;
      }
    }
}
```

三、问答题

1. 分析 GridView、DataList 和 Repeater 三个控件的特点(分析各自的功能、效率，并说明在哪种情况下使用)。

2. 简单介绍 GridView 控件，并举例说明 GridView 控件的使用方法。

3. 简述 ListView 控件及该控件如何显示和编辑数据。

4. 比较 GridView、DetailView、FormView 和 ListView 控件的使用。

单元9 数据绑定控件应用进阶

教学目标：

- 掌握获取 GridView 单元格数据的相关操作。
- 掌握 GridView 基于单元格的更新的方法。
- 掌握 GridView 常用事件的处理。
- 掌握 GridView 中如何删除数据。
- 掌握其他数据绑定控件与 GridView 相类似的操作。

9.1 知识准备

9.1.1 获取 GridView 单元格数据

上一单元完成了新知书店管理端图书的更新和添加功能。更新数据时，为管理端添加了图书详细信息页面从而实现了更新功能，除此方法之外，本单元还将介绍另外一种实现更新操作的方法——基于单元格数据的更新(以 GridView 控件为例)。在前面的单元中，我们学习了 GridView 的数据绑定以及数据分页等技巧，GridView 是一个功能及其丰富的控件，本单元将深入介绍 GridView 的其他常用操作，如在 GridView 中实现单选、多选删除数据。使用 GridView 控件操作数据时，常常需要获取 GridView 单元格数据，例如使用详细信息页面更新数据时，就可以通过在列表页面获取到当前记录的主键 ID，再将该 ID 值传递到详细页面实现数据绑定以及更新操作，本单元将介绍通过行、列或通过 DataKey 属性两种获取单元格数据的方式。

1. 通过行、列获取单元格数据

我们曾介绍过使用 GridView 的 BoundField 和 TemplateField 等七种数据绑定列类型来控制 GridView 中某一列的显示。此外，也可以通过 GridView 的行来操作 GridView 中的数据。在获取 GridView 单元格数据前，首先来分析 GridView 的结构，GridView 的结构如图 9-1 所示。

从图 9-1 中可以看出，GridView 同 WinForms 中的数据绑定控件 DataGridView 一样，也是由多个行(GridViewRow)组成的，形式上类似于 Table 中的 TR，DataTable 中的 DataRow。而 GridViewRow 是由一个个单元格组成的，GridView 中单元格的类型是 TableCell。

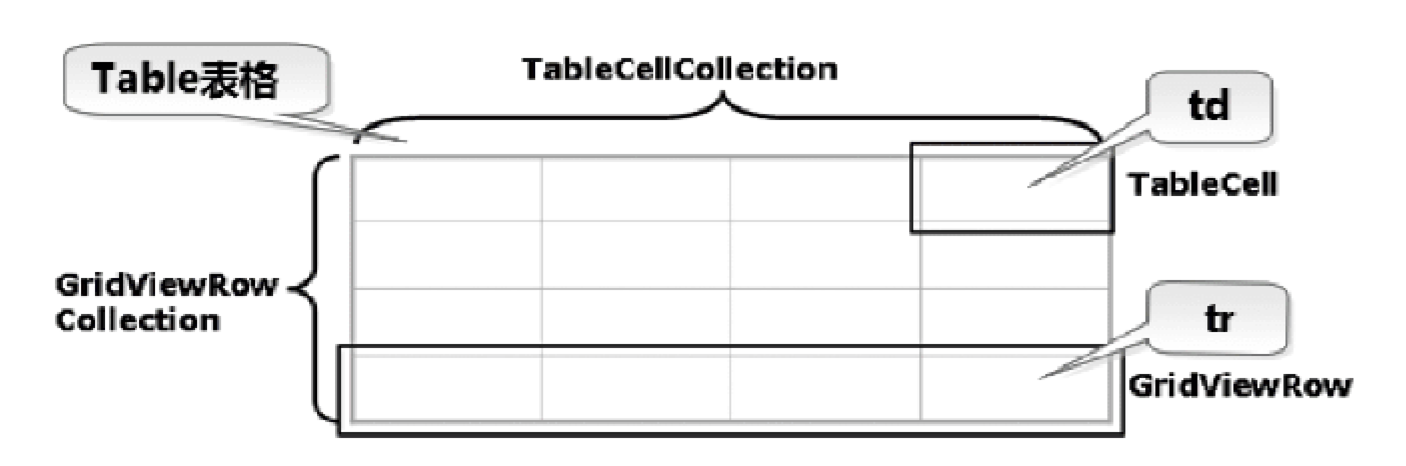


图 9-1　GridView 的结构图

GridView 的 GridViewRow 是一个集合，可以通过 gvHR. Rows[index]获得一个 GridViewRow 对象(其中 gvHR 为 GridView 控件对象名)，同理，GridViewRow 中的单元格也是一个集合，可以通过 this. gvHR. Rows[index]. Cell[0]. Text 获取表中第 index+1 行第一列的数据，也可以使用以下代码取得相同的数据。

```
GridViewRow gr = gvHR.Rows[rowIndex];
string text = gr.Cells[0].Text;
```

由此可以归纳出通过 GridView 中的行和列取得单元格数据的语法如下。

```
GridView 控件 ID.Rows[rowIndex].Cell[columnIndex].Text
```

或

```
GridViewRow gr = GridView 控件 ID.Rows[rowIndex];
string text = gr.Cells[columnIndex].Text;
```

2. 通过 DataKeys 属性获取单元格数据

上面介绍的通过 GridView 的行和列获取单元格数据的问题在于要将获取的数据显示在 GridView 中，而实际项目开发中获取的数据往往是数据表里的主键，对于用户来说应该是不可见的，那么如何在不显示主键的情况下获取当前选择记录的主键呢?

有人可能已经想到把该主键的 Visible 属性设置为 false，然后再用 GridView 的行和列来获取主键，但是问题在于如果将某列的 Visible 属性设置为 false，则该列的数据将不会往返于服务器端和客户端，因此不能用这个方法直接获取被隐藏单元格的值。

要实现此功能要用到 GridView 的两个属性 DataKeyNames 和 DataKeys，这两个属性在介绍 GridView 时曾提到过。

- DataKeyNames：用来获取或指定 GridView 中主键字段名称的数组，多个主键字段之间使用逗号隔开。
- DataKeys：用来获取 GridView 中使用 DataKeyNames 设置的每一行主键值的对象集合。

认识了 DataKeyNames 和 DataKeys 属性，如何使用它们来获取 GridView 中的数据?

1) 设置绑定主键

设置绑定主键有如下两种方式：

- 在视图设计器中设置 DataKeyNames 属性，多个主键字段使用逗号分隔。
- 通过如下代码设置绑定主键：

```
GridView 控件 ID.DataKeyNames = new string[] { "主键字段名称 1","主键字段名称 2",…};
GridView 控件 ID.DataBind();
```

2）获取主键值

获取单个主键的代码如下。

```
GridView 控件 ID.DataKeys[index].Value.ToString();
```

获取多个主键的代码如下。

```
GridView 控件 ID.DataKeys[index]["主键字段名称 1"].ToString();
```

或

```
GridView 控件 ID. DataKeys[index].Values["主键字段名称 1"].ToString();
```

实现员工详细信息更新，获取单元格数据功能，使用 DataKeys 属性方式实现的代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        //省略其他的绑定代码
        //设置主键字段为 userid
        this.gvHR.DataKeyNames = new string[] { "userid"};
        this.gvHR.DataBind();
    }
}

protected void gvHR_SelectedIndexChanging(object sender, GridViewSelectEventArgs e)
{
    int index = e.NewSelectedIndex;
    //通过 DataKey 获取行主键值
    string key = this.gvHR.DataKeys[index].Value.ToString();
    Response.Redirect("modify.aspx?id=" + key);
}
```

页面的 Page_Load 事件方法内设置 DataKeyNames 属性的代码也可以通过如图 9-2 所示来设置。

9.1.2 基于单元格的更新

除了在详细页面可以实现某一条记录的更新外，还可以选择使用在 GridView 的单元

格中实现更新功能，即基于单元格的更新。基于单元格更新的基本思路是先使要更新的行处于编辑模式，然后在该行上直接修改，修改后，将数据更新到数据库，最后重新绑定 GridView。下面按这个思路现实员工详细信息更新功能。



图 9-2　设置 GridView 的 DataKeyNames 属性

1. 设置模板列

当某一行处于编辑状态时，该行需提供 TextBox、DropDownList 等控件供用户输入或选择。编辑状态中的 TextBox 等控件是如何添加的？可以借助编辑模板列 EditTemplate 来实现，在单元 8 中实现“全选”功能时，使用过 ItemTemplate 和 HeaderTemplate，分别在普通项和标题头添加了一个复选框。ItemTemplate 提供用于显示信息的模板，而 EditTemplate 提供用于编辑信息的模板，为显示员工信息的 GridView 添加编辑模板的 Default.aspx 页面代码如下所示。

```
<asp:GridView ID="gvHR" runat="server" AutoGenerateColumns="False" DataKeyNames="userId"
              OnRowEditing="gvHR_RowEditing"   OnRowUpdating="gvHR_RowUpdating"
              OnRowCancelingEdit="gvHR_RowCancelingEdit">
    <Columns>
        <asp:TemplateField HeaderText="姓名">
            <ItemTemplate>
                <asp:Label ID="lblName" runat="server" Text='<%#Eval("username") %>'>
</asp:Label>
            </ItemTemplate>
            <EditItemTemplate>
                <asp:TextBox ID="txtName" runat="server" Text='<%#Bind("username") %>'>
                </asp:TextBox>
            </EditItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="性别">
            <ItemTemplate>
                //Eval()方法的返回类型是 object,这里需要先进行类型转换,再使用三元运算符比较
                <asp:Label ID="lblGender" runat="server"
                    Text='<%#Eval("gender").ToString()=="True"?"男":"女"%>'></asp:Label>
            </ItemTemplate>
            <EditItemTemplate>
                <asp:DropDownList ID="ddlGender" runat="server">
                    <asp:ListItem Value="True" Text="男"></asp:ListItem>
                    <asp:ListItem Value="False" Text="女"></asp:ListItem>
                </asp:DropDownList>
            </EditItemTemplate>
        </asp:TemplateField>
```

```
        <asp:TemplateField HeaderText = "职务">
            <ItemTemplate>
                //GetDutyName()方法是后置代码中(服务器端)的一个方法,用于根据职务编号获
                //取职务名称
                <asp:Label ID = "lblDuty" runat = "server"
                    Text = '<%#GetDutyName(Eval("postID").ToString()) %>'></asp:Label>
            </ItemTemplate>
            <EditItemTemplate>
                <asp:DropDownList ID = "ddlDuty" runat = "server">
                </asp:DropDownList>
            </EditItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText = "电话">
            <ItemTemplate>
            <asp:Label ID = "lblPhone" runat = "server" Text = '<%#Eval("telePhone") %>'></
asp:Label>
            </ItemTemplate>
            <EditItemTemplate>
            <asp:TextBox ID = "txtPhone" runat = "server" Text = '<%#Bind("telePhone") %>'>
</asp:TextBox>
            </EditItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText = "地址">
            <ItemTemplate>
                <asp:Label ID = "lblAddress" runat = "server" Text = '<%#Eval("address") %>'>
</asp:Label>
            </ItemTemplate>
            <EditItemTemplate>
            <asp:TextBox ID = "txtAddress" runat = "server" Text = '<%#Bind("address") %>'>
</asp:TextBox>
           </EditItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

Default.aspx 页面后置 cs 文件的部分代码如下。

```
protected string GetDutyName(string dutyId)
{
    StringBuilder sbSql = new StringBuilder();
    sbSql.AppendLine("SELECT");
    sbSql.AppendLine("     [postName] ");
    sbSql.AppendLine("FROM ");
    sbSql.AppendLine("     [post] ");
    sbSql.AppendLine("WHERE ");
    sbSql.AppendLine("     [postId] = @postId");
    SqlParameter[] para = new SqlParameter[] { new SqlParameter("@postId", dutyId) };
    DataTable dt = DBHelper.GetDataSet(sbSql.ToString (),para);
    return dt.Rows[0][0].ToString();
}
```

“姓名”、“电话”和“地址”列需要用户手动输入信息，因此在 EditItemTemplate 中添加的是 TextBox 控件，注意添加的 TextBox 需要使用 Eval()方法或者 Bind()方法实现编辑状态下的数据绑定。“性别”和“职务”列为避免用户的错误输入，提供 DropDownList 下拉列表供用户选择。“性别”只有“男”和“女”，可以选择静态方式添加数据，“职务”列中的数据要根据数据库动态更新，这里采用在后置代码中动态绑定方式。

除了使用手写代码配置配置模板列，还可以通过“GridView 任务”窗口进行可视化地编辑模板。GridView 提供的模板编辑模式可以通过控件的拖放、属性设置同样实现模板配置。这种方式虽然简单，但对于初学者而言推荐手写方式实现模板编辑。

2. 添加命令按钮

仅仅提供了编辑状态下的输入、列表选择控件还不能实现编辑功能，还需要添加命令按钮来触发编辑模式下的时间，前面介绍过 CommandField，它与 ButtonField 类似，除了创建命令按钮外，它还提供了在数据绑定控件中执行选择、编辑、插入或删除操作。

这里为 GridView 添加“编辑、更新、取消”命令按钮，并设置该列的标题为“操作”，编辑按钮默认是超链接形式，还可以把它更改为 Button、Image 形式，只要在设计器里设置即可，如图 9-3 所示。



图 9-3　添加并设置命令按钮

单击图 9-3 中的“确定”按钮后，查看源视图，代码中添加了如下代码。

```
<asp:CommandField HeaderText = "操作" ShowEditButton = "True" />
```

运行页面 Default.aspx，效果如图 9-4 所示。

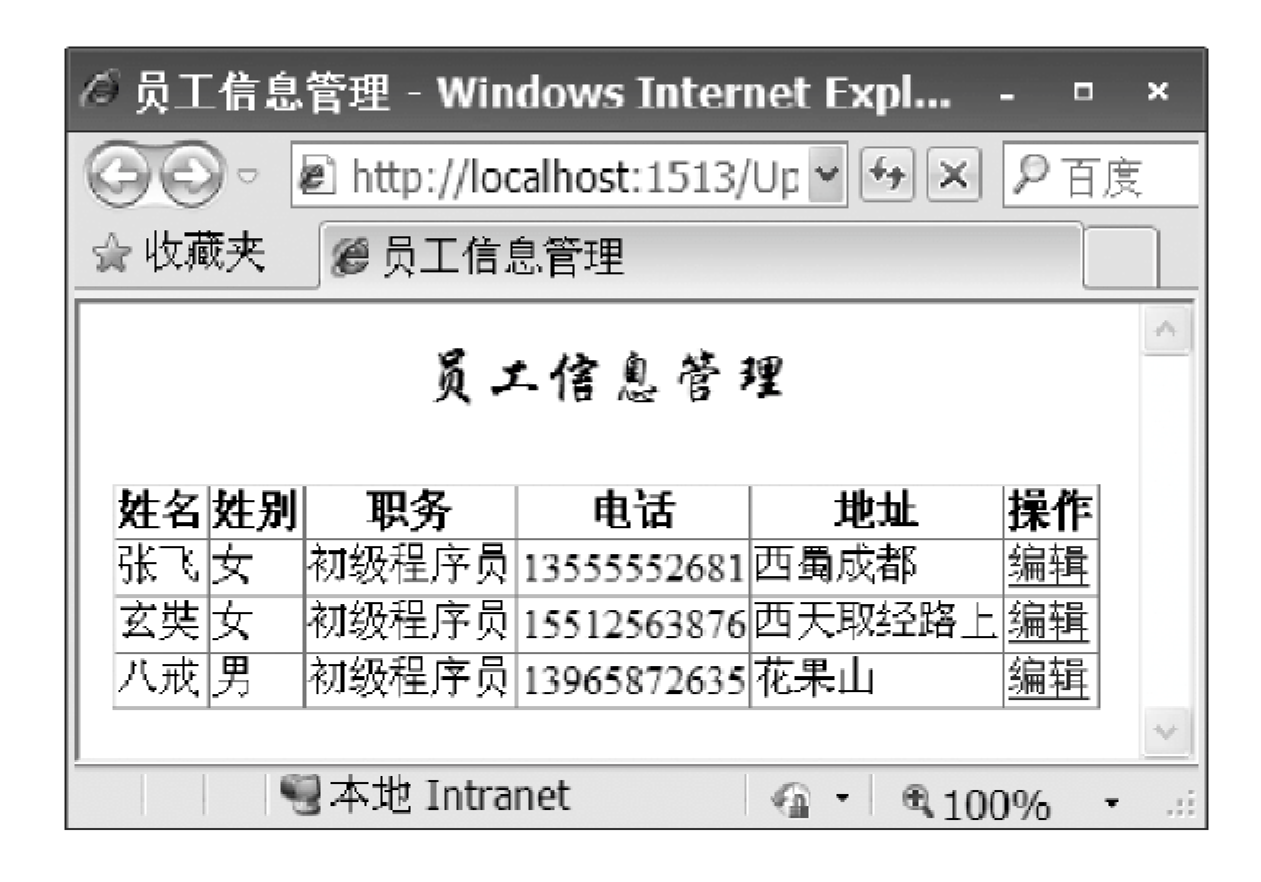


图 9-4　添加“编辑”按钮后的页面运行效果

3. 处理事件

1）RowEditing 事件

单击图 9-4 中任意一条记录的“编辑”按钮，出现如图 9-5 所示的错误页面。

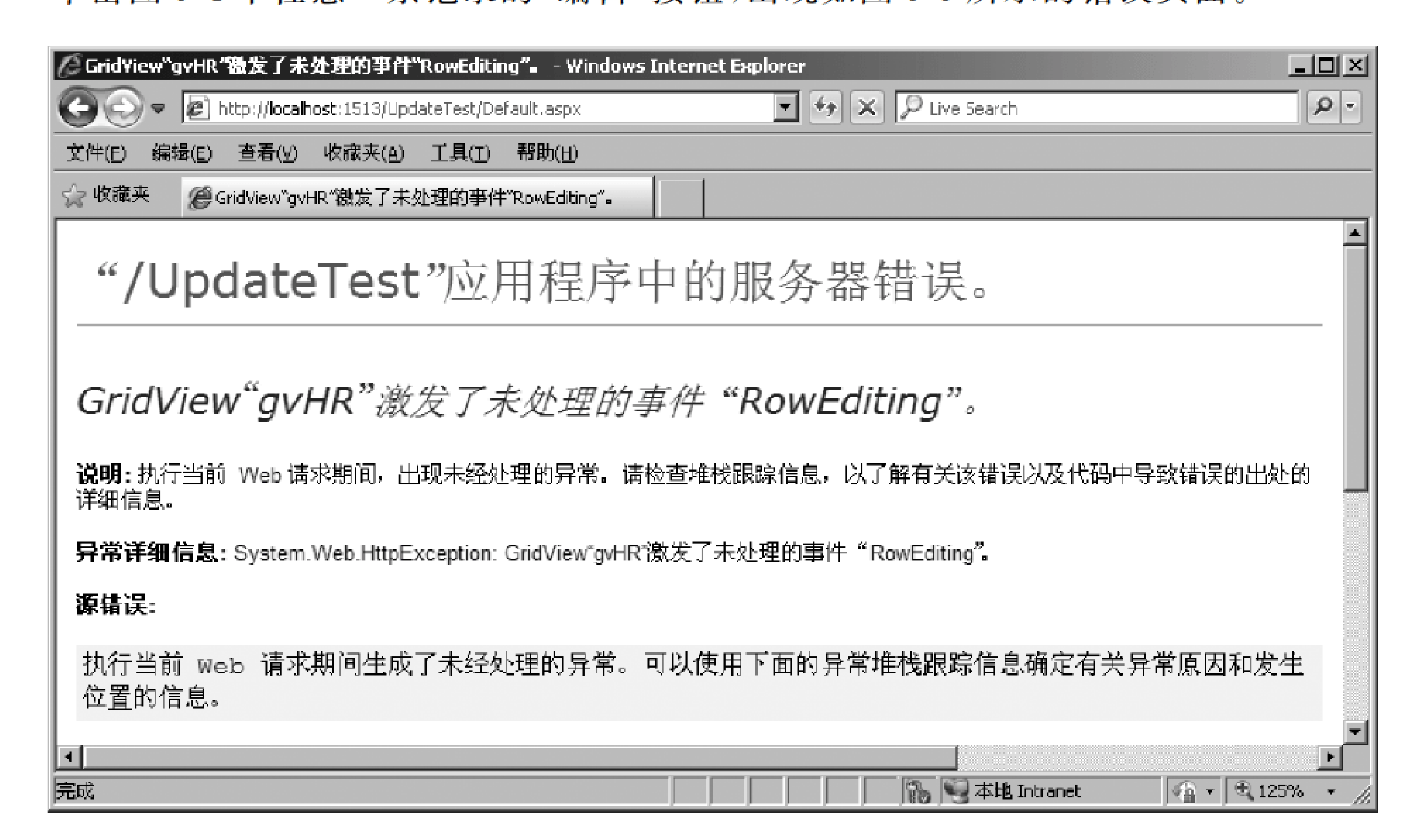


图 9-5　未处理“RowEditing 事件”，单击“编辑”按钮效果

通过错误信息提示很容易分析出错误原因：“编辑”按钮触发了 RowEditing 事件，而 RowEditing 事件并没有处理。要解决这个错误，只要编码处理该事件即可。处理该事件之前，先来了解一下 RowEditing 事件。单击某一行的“编辑”按钮后，在 GridView 控件进入编辑模式之前，将引发该事件。RowEditing 事件带有两个参数，类型分别是 object 和 GridViewEditEventArgs。其中 GridViewEditEventArgs 对象属性如下。

- NewEditIndex：获取或设置所编辑的行的索引。

- Cancel：将 Cancel 属性设置为 true，可以取消编辑操作。

处理 RowEditing 事件完成员工信息的编辑功能，页面后置代码如下所示。

```
/// <summary>
/// gvHR 的 RowEditing 事件处理方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void gvHR_RowEditing(object sender, GridViewEditEventArgs e)
{
    try
    {
        this.gvHR.EditIndex = e.NewEditIndex;      //设置编辑行索引
        HRDataBind();                              //由于回传，页面无法保存对象的状态
                                                   //需重新绑定
    }
    catch (Exception)
    {
         Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "info", "<script>
alert('操作失败!')</script>");
    }
}

/// <summary>
/// 员工信息绑定方法
/// </summary>
protected void HRDataBind()
{
    StringBuilder sbSql = new StringBuilder();
    sbSql.AppendLine("SELECT");
    sbSql.AppendLine("      * ");
    sbSql.AppendLine("FROM ");
    sbSql.AppendLine("     [users] ");

    DataTable dt = DBHelper.GetDataSet(sbSql.ToString());
    this.gvHR.DataSource = dt;
    this.gvHR.DataBind();
}
```

代码中使用了 GridView 的 EditIndex 属性，该属性可以获取或设置要编辑的行的索引，当该属性被设置为某一行的索引时，该行进入编辑状态。要编辑的行的索引从 0 开始，默认值为−1，表示行没有正在被编辑。

由于“职务”列没有设置绑定字段，其内容为空。需要在服务器端编码填充“职务”列，并且实现将编辑前某行的“性别”和“职务”信息带到编辑状态，显示到对应列中。

为了实现在编辑状态中默认显示编辑前的“性别”和“职务”信息，可以通过在 EditItemTemplate 中添加一个隐藏控件，使用该隐藏控件的 Value 或 Text 等属性绑定性别编号、职务编号，这样编辑状态中就可以找到该控件，取得 Value 或 Text 等属性值赋给

DropDownList 的 SelectedValue 即可。对于“职务”的绑定，只要找到需要编辑行的“职务”列的 DropDownList 完成绑定操作即可，编辑行的索引通过 GridViewEditEventArgs 对象的 NewEditIndex 属性得到。

在 ASP.NET，HiddenField 控件用来存储非显示值的隐藏字段，其 value 属性可用来获取或设置隐藏字段的值。在 GridView 中可以通过如下语法得到某一行的控件。

```
GridView 控件 ID.Rows[index].FindControl(要得到的服务器控件 ID)
```

要实现员工信息中“职务”列数据绑定和“性别”默认信息显示功能，代码如下所示。

页面 Default.aspx 中的代码如下。

```
<asp:templatefield headertext="性别">
    <%--省略其他代码--%>
    <EditItemTemplate>
        <asp:HiddenField ID="hfGenderId" runat="server" Value='<%#Eval("gender") %>'/>
        <asp:DropDownList ID="ddlGender" runat="server">
            <asp:ListItem Value="True" Text="男"></asp:ListItem>
            <asp:ListItem Value="False" Text="女"></asp:ListItem>
        </asp:DropDownList>
    </EditItemTemplate>
</asp:templatefield>
<asp:templatefield headertext="职务">
    <%--省略其他代码--%>
    <EditItemTemplate>
        <asp:HiddenField ID="hfDuty" runat="server" Value='<%#Eval("postID") %>'/>
        <asp:DropDownList ID="ddlDuty" runat="server">
        </asp:DropDownList>
    </EditItemTemplate>
</asp:templatefield>
```

页面 Default.aspx 后置 cs 文件相应方法代码如下。

```
/// <summary>
/// "编辑"按钮事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void gvHR_RowEditing(object sender, GridViewEditEventArgs e)
{
    try
    {
        this.gvHR.EditIndex = e.NewEditIndex;  //设置编辑行索引
        HRDataBind();                          //由于回传,页面无法保存对象的状态,需重新绑定
        GetEditStateGender(e.NewEditIndex);//获取并设置"性别"信息的方法
        GetEditStateDuty(e.NewEditIndex);  //获取并设置"职务"信息的方法
    }
    catch (Exception)
    {
```

```
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "info", "<script>
alert('操作失败!')</script>");
        }
    }

    /// <summary>
    /// 根据索引绑定编辑状态下的"性别"
    /// </summary>
    /// <param name="index"></param>
    public void GetEditStateGender(int index)
    {
        //通过行索引找到绑定"性别编号"的 HiddenField 控件
        HiddenField hfGenderItem = this.gvHR.Rows[index].FindControl("hfGenderId") as
HiddenField;
        DropDownList ddlGenderItem = this.gvHR.Rows[index].FindControl("ddlGender") as
DropDownList;
        //设置"性别"下拉列表框的默认值
        ddlGenderItem.SelectedValue = hfGenderItem.Value;
    }

    /// <summary>
    /// 根据索引绑定编辑状态下的"职务"列表
    /// </summary>
    /// <param name="index"></param>
    public void GetEditStateDuty(int index)
    {
        //通过行索引找到"职务"下拉列表框
        DropDownList ddlDutyItem = this.gvHR.Rows[index].FindControl("ddlDuty") as DropDownList;

        StringBuilder sbSql = new StringBuilder();
        sbSql.AppendLine("SELECT");
        sbSql.AppendLine("      * ");
        sbSql.AppendLine("FROM ");
        sbSql.AppendLine("      [post] ");
        DataTable dt = DBHelper.GetDataSet(sbSql.ToString());

        ddlDutyItem.DataSource = dt;
        ddlDutyItem.DataValueField = "postId";
        ddlDutyItem.DataTextField = "postName";
        ddlDutyItem.DataBind();
        //通过行索引找到"职务编号"的 HiddenField 控件
        HiddenField hfDutyItem = this.gvHR.Rows[index].FindControl("hfDuty") as HiddenField;
        //设置"职务"下拉列表框的默认值
        ddlDutyItem.SelectedValue = hfDutyItem.Value;
    }
```

2) RowUpdating 事件

在编辑状态，修改数据后单击“更新”按钮，同样会提示“gvHR 激发了为处理事件 RowUpdating”的错误。单击某一行的“更新”按钮以后，在 GridView 控件对该行进行更新

之前，引发该事件。下面继续完成“员工信息”的更新功能，现实如图 9-6 所示效果。



图 9-6 基于单元格更新员工信息的效果

实现更新功能需要获取当前编辑行的信息，并将该信息更新到数据库。更新完成后可以使用注册脚本方式提示更新成功，并结束当前的编辑状态，代码如下所示。

```
/// <summary>
/// gvHR 的 RowUpdating 事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void gvHR_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    try
    {
        users user = GetInputData(e.RowIndex);   //取得编辑行修改后的数据
        UpdateUsers(user);                       //更新数据库方法
         Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "info", "<script>
alert('更新成功!')</script>");
        this.gvHR.EditIndex = -1;                //退出编辑状态
        HRDataBind();                            //重新绑定
    }
    catch (Exception)
    {
         Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "info", "<script>
alert('操作失败!')</script>");;
    }
}

/// <summary>
/// 取得编辑行修改后的数据
/// </summary>
/// <param name="index"></param>
/// <returns></returns>
public users GetInputData(int index)
{
```

```
        string userId = this.gvHR.DataKeys[index].Value.ToString();
        string username = (this.gvHR.Rows[index].FindControl("txtName") as TextBox).Text;
        string gender = (this.gvHR.Rows[index].FindControl("ddlGender") as DropDownList).
    SelectedValue.ToString();
        string postID = (this.gvHR.Rows[index].FindControl("ddlDuty") as DropDownList).
    SelectedValue.ToString();
        string telePhone = (this.gvHR.Rows[index].FindControl("txtPhone") as TextBox).Text;
        string address = (this.gvHR.Rows[index].FindControl("txtAddress") as TextBox).Text;
        users user = new users();
        user.userId = userId;
        user.username = username;
        user.gender = gender;
        user.postID = postID;
        user.telePhone = telePhone;
        user.address = address;
        return user;
    }

    /// <summary>
    /// 更新数据库方法
    /// </summary>
    /// <param name="user"></param>
    public void UpdateUsers(users user)
    {
        StringBuilder sbSql = new StringBuilder();
        sbSql.AppendLine("UPDATE ");
        sbSql.AppendLine("    [users]");
        sbSql.AppendLine("SET");
        sbSql.AppendLine("    [username] = @username,");
        sbSql.AppendLine("    [gender] = @gender,");
        sbSql.AppendLine("    [postID] = @postID,");
        sbSql.AppendLine("    [telePhone] = @telePhone,");
        sbSql.AppendLine("    [address] = @address");
        sbSql.AppendLine("WHERE");
        sbSql.AppendLine("    [userId] = @userId");
        SqlParameter[] para = new SqlParameter[] {
                                    new SqlParameter ("@userId",user.userId),
                                    new SqlParameter ("@username",user.username),
                                    new SqlParameter ("@gender",user.gender),
                                    new SqlParameter ("@postID",user.postID),
                                    new SqlParameter ("@telePhone",user.telePhone),
                                    new SqlParameter ("@address",user.address)
                                };
        DBHelper.ExecuteCommand(sbSql.ToString(), para);
    }
```

3）RowCancelingEdit 事件

在编辑状态，除了“更新”按钮外，还有“取消”按钮，供用户取消当前编辑状态，“取消”按钮触发的是 RowCancelingEdit 事件事件。单击编辑模式中某一行的“取消”按钮以后，在该

行推出编辑模式之前引发该事件,实现员工信息编辑行取消操作的代码如下所示。

```
/// <summary>
/// gvHR 的 RowCancelingEdit 事件方法
/// </summary>
/// <param name = "sender"></param>
/// <param name = "e"></param>
protected void gvHR_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e)
{
    try
    {
        this.gvHR.EditIndex = -1;     //退出编辑状态
        HRDataBind();                 //重新绑定
    }
    catch (Exception)
    {
         Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "info", "<script>
alert('操作失败!')</script>");;
    }
}
```

注意:在详细页面更新数据的方式适合 GridView 的单元格里的信息量较大,或者有字段没有在单元格里显示的情形。比如新知书店的书籍内容简介,内容较多,它本身不显示在 GridView 中,所以在编辑时需要连接到另一个页面进行修改;基于单元格的修改只适用于信息量较少且在 GridView 中显示时不缺少修改的字段。

9.1.3 RowCommand 事件

在员工信息管理系统中,当单击 CommandField 中的"编辑"按钮会引发 RowEditing 事件,当单击"更新"按钮时会引发 RowUpdating 事件,事实上单击这些按钮时都会首先引发 RowCommand 事件。RowCommand 事件发生在单击 GridView 控件中的任意一个时。可是如果 GridView 中的多个模板列都添加了按钮控件,那么如何在一个 RowCommand 事件中区分是哪一列的按钮被单击了呢?RowCommand 事件有两个参数:一个是 object 类型,另一个类型是 GridViewEditEventArgs。使用 GridViewEditEventArgs 类型的如下两个属性就可以区分哪一列的按钮被单击了。

- CommandArgument:引发事件源的命令参数。
- CommandName:引发事件源的命令名称。

通常使用 CommandArgument 设置或获取按钮所带的参数信息,这样能区分具体的按钮控件。

若要执行一些常见的增删改查操作,可以使用 GridView 的一些内置命令,使用时只需将 CommandName 属性设置成表 9-1 中的某个值。

提示:图 9-3 中添加的 CommandField 的"编辑"、"更新"、"取消"命令按钮就是使用了 GridView 内置的 Edit、Update 和 Cancel 命令,从而引发相应的事件。

表 9-1　GridView 的一些内置命令

CommandName 值	说　明
Cancel	退出编辑模式，引发 RowCancelingEdit 事件
Delete	删除当前记录，引发 RowDeleting 和 RowDeleted 事件
Edit	将当前记录处于编辑模式，引发 RowEditing 事件
Select	选择当前记录，引发 Row_Selecting 和 Row_Selected 事件
Update	更新数据源中的当前记录，引发 RowUpdating 和 RowUpdated 事件

我们已经学习了使用 GridView 完成数据格式化显示和更新操作，下面介绍 GridView 中的另外一种常用操作——删除。

删除数据的一般思路如下。

(1)用户单击“删除”按钮，考虑到用户体验，应弹出确认删除的提示。

(2) 获取删除行的主键。

(3) 执行删除操作，删除后重新绑定数据。

删除 GridView 中的数据可以采用两种方式：多选删除和单选删除。

1. 单选删除

我们以员工信息管理系统为例，介绍单选删除的实现过程。单击图 9-7 中某条记录的“删除”按钮，弹出“确认删除吗?”的提示框，单击“确定”按钮，则执行对当前行的删除操作；单击“取消”按钮，则取消当前删除操作。



图 9-7　单击某一条记录的“删除”按钮效果

下面分步骤介绍其实现过程。

1) 添加“删除”按钮

为 GridView 添加模板列，并在 ItemTemplate 中添加 LinkButton(当然也可以添加 Button、为了与“编辑”按钮样式匹配，此处添加 LinkButton)。然后将 LinkButton 的

CommandName 设置为内置命令 Delete，CommandArgument 设置为绑定用户编号，这样就可以在服务器端取得当前删除行的用户编号，代码如下。

```
<asp:TemplateField  HeaderText = "删除">
<ItemTemplate>
    <asp:LinkButton ID = "lnkbtnDel" runat = "server" Text = "删除"
          CommandArgument  = '<%#Eval("userId") %>'
          CommandName  = "Delete"></asp:LinkButton>
</ItemTemplate>
</asp:TemplateField>
```

2）给“删除”按钮注册 JavaScript 脚本

需要为 GridView 中每一数据行的“删除”按钮添加 JavaScript 脚本，可以选择在 GridView 控件呈现之前进行添加，这就需要用到 RowDataBound 事件，该事件将 GridView 中的数据行绑定到数据时引发。为“删除”按钮添加“确认删除吗?”的友好提示的代码如下所示。

```
protected void gvHR_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        LinkButton lnkbtnDel  =  e.Row.FindControl("lnkbtnDel") as LinkButton;
        lnkbtnDel.Attributes.Add("onclick", "return confirm('确认删除吗?')");
    }
}
```

为了区分执行的是数据行还是标题行、脚注行等 UI 元素行，使用 DataControlRowType 枚举类型进行判断，只有当显示的数据行时才注册 JavaScript 脚本。

3）处理 RowDeleting 事件

由于将 LinkButton 的 CommandName 设置为 GridView 的内置命令 Delete，因此当单击“删除”按钮后将引发 RowDeleting 事件，只需要在该事件中获取删除行的用户编号，执行删除操作即可，代码如下所示。

```
/// <summary>
///  gvHR 的 RowDeleting 事件方法
/// </summary>
/// <param name = "sender"></param>
/// <param name = "e"></param>
protected void gvHR_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    try
    {
        //找到当前删除行 Id 为 lnkbtnDel 的 LinkButton 控件
        LinkButton lnkbtnDel  =  gvHR.Rows[e.RowIndex].FindControl("lnkbtnDel") as LinkButton;
        //执行删除操作
        StringBuilder sbSql  =  new StringBuilder();
```

```
            sbSql.AppendLine("DELETE FROM ");
            sbSql.AppendLine("    [users]");
            sbSql.AppendLine("WHERE");
            sbSql.AppendLine("    [userid] = @userid");
            //取得删除行绑定的用户编号
            SqlParameter[] para = new SqlParameter[]
                                  {
                                  new SqlParameter("@userId", lnkbtnDel.CommandArgument.ToString())
                                  };
            DBHelper.ExecuteCommand(sbSql.ToString(), para);
            HRDataBind();            //重新绑定
        }
        catch (Exception ex)
        {
            Page.ClientScript.RegisterClientScriptBlock(this.GetType(), "",
                                  "<script>alert('" + ex.Message + "')</script>");
        }
    }
```

至此已经完成了 GridView 中的单行删除操作。如果不想使用 GridView 内置的命令，使用自定义的命令也可以完成删除操作，但需要将按钮的 CommandName 设置成除 GridView 内置的命令值以外的其他字符串，并处理 RowCommand 事件，例如设置成 Del，代码如下所示。

页面 GridViweDelDemo.aspx 中的代码如下。

```
<asp:TemplateField HeaderText = "删除">
    <ItemTemplate>
        <asp:LinkButton ID = "lnkbtnDel" runat = "server" Text = "删除"
            CommandArgument = '<%# Eval("userId") %>'
            CommandName = "Del">
        </asp:LinkButton>
    </ItemTemplate>
</asp:TemplateField>
```

页面 GridViweDelDemo.aspx 后置 cs 文件中 RowCommand 事件方法代码如下：

```
/// <summary>
/// gvHR 的 RowCommand 事件方法
/// </summary>
/// <param name = "sender"></param>
/// <param name = "e"></param>
protected void gvHR_RowCommand(object sender, GridViewCommandEventArgs e)
{
    if (e.CommandName == "Del")    //通过"Del"判断单击的是否为"删除"列
    {
        //执行删除操作
        StringBuilder sbSql = new StringBuilder();
        sbSql.AppendLine("DELETE FROM ");
```

```
            sbSql.AppendLine("     [users]");
            sbSql.AppendLine("WHERE");
            sbSql.AppendLine("     [userid] = @userid");
            //取得删除行绑定的用户编号
            SqlParameter[] para = new SqlParameter[] {
                                  new SqlParameter("@userId", e.CommandArgument.ToString())
                                          };
            DBHelper.ExecuteCommand(sbSql.ToString(), para);
        }
        HRDataBind();              //重新绑定
    }
```

2. 多行删除

多行删除即选中 GridView 中的多条记录后，单击“删除”按钮后对 GridView 中选中的记录执行删除操作，效果如图 9-8 所示。



图 9-8　员工信息系统多选删除效果

提示：“删除”按钮的友好提示框可以采用在页面加载时为“删除”按钮注册脚本方式。如果确认删除，则循环 GridView 中的每一条记录并进行判断。如果当前行是选择状态，则执行删除操作。可以借助前面单元中学习的全选功能实现删除操作。

页面 Default.aspx 后置 cs 文件的代码如下。

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
      //其他代码省略
      this.btnDel.Attributes.Add("onclick", "return confirm('确认删除吗?')");
    }
}
```

```
/// <summary>
/// "删除"按钮按下的处理事件方法
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void btnDel_Click(object sender, EventArgs e)
{
    foreach (GridViewRow gr in this.gvHR.Rows)//循环每一行
    {
        //获取模板列中的CheckBox控件对象
        CheckBox chk = gr.FindControl("chbSelect") as CheckBox;
        if (chk.Checked)
        {
            string key = this.gvHR.DataKeys[gr.RowIndex].Value.ToString(); //获取主键
            //执行删除操作
            StringBuilder sbSql = new StringBuilder();
            sbSql.AppendLine("DELETE FROM ");
            sbSql.AppendLine("    [users]");
            sbSql.AppendLine("WHERE");
            sbSql.AppendLine("    [userid] = @userid");
            SqlParameter[] para = new SqlParameter[] { new SqlParameter("@userId", key) };
            DBHelper.ExecuteCommand(sbSql.ToString(), para);
        }
    }
    HRDataBind();
}
```

提示：GridView 中 CommandField 的“删除”按钮也可实现删除功能，但在执行删除前没有提示信息，所以推荐使用 Templateield 完成删除操作。

如果要删除数据。往往要考虑数据库中是否存在与之关联的其他表数据，这些关联数据是否也要一起删除。所以做数据库设计的时候一般都考虑用外键约束、触发器来避免删除或更新带来的“脏”数据。

9.2 单元任务

任务 9-2-1 实现“新知书店”会员状态显示及管理功能

【任务描述】

实现如图 9-9 所示的会员状态显示及管理功能，当页面加载时分页显示所有会员信息，实现全选功能。

【任务实施】

1. 会员状态管理数据访问层与业务逻辑层的实现

页面加载时，要获取所有会员信息，数据访问层和业务逻辑层要能获取所有会员信息，



图 9-9　管理员端的会员状态管理页面

会员有“正常”和“无效”两种状态，在对某会员进行状态管理时，能实现根据会员 Id 值对用户表 Users 里相应用户记录进行修改。

1）会员状态管理数据访问层的实现

在 BookShopDAL 项目的 UserService.cs 类中，添加获取所有会员信息的 GetUsers() 方法，并返回一个用户列表对象集合，代码如下：

```
/// <summary>
/// 查询所有用户
/// </summary>
/// <returns></returns>
public  List<User> GetUsers()
{
    string sqlAll = "SELECT * FROM Users";
    return GetUsersBySql(sqlAll);
}

/// <summary>
/// 依据 SQL 语句查询用户
/// </summary>
/// <param name="safeSql"></param>
/// <returns></returns>
private  List<User> GetUsersBySql(string safeSql)
{
    List<User> list = new List<User>();
    DataSet ds = SqlHelper.ExecuteDataset(this.connection, CommandType.Text, safeSql);
    if (ds.Tables.Count > 0)
    {
        DataTable dt = ds.Tables[0];
        foreach (DataRow row in dt.Rows)
        {
```

```
            User user = new User();
            user.Id = (int)row["Id"];
            user.LoginId = (string)row["LoginId"];
            user.LoginPwd = (string)row["LoginPwd"];
            user.Name = (string)row["Name"];
            user.Address = (string)row["Address"];
            user.Phone = (string)row["Phone"];
            user.Mail = (string)row["Mail"];
            user. UserState = new  UserStateService ( ). GetUserStateById (( int ) row
["UserStateId"]);        //FK
            user. UserRole  = new  UserRoleService ( ). GetUserRoleById (( int ) row
["UserRoleId"]);        //FK
            list.Add(user);
        }
    }
    return list;
}
```

添加一个根据当前会员 Id 和状态进行修改的方法，代码如下：

```
/// <summary>
/// 更改会员状态
/// </summary>
/// <param name = "id">会员 Id</param>
/// <param name = "status">会员状态</param>
public  bool ModifyUserState(int id, UserStates state)
{
    string sql = "Update users SET userstateid = " + Convert.ToByte(state) + " WHERE Id =
@UserId";
    return  SqlHelper. ExecuteNonQuery ( this. connection,  CommandType. Text,  sql,  new
SqlParameter("@UserId", id)) > 0;
}
```

2）会员状态管理业务逻辑层的实现

在 BookShopBLL 项目的类文件 UserManager.cs 中，添加代码如下：

```
public List<User> GetUsers()
{
    return new UserService().GetUsers();
}

/// <summary>
/// 根据 id 修改用户状态
/// </summary>
/// <param name = "userId"></param>
public bool ModifyUserState(int userId)
{

    User user = new UserService().GetUserById(userId);
```

```
        UserStates state = UserStates.正常;
        if (user.UserState.Id == Convert.ToInt32(UserStates.正常))
        {
            state = UserStates.无效;
        }
        else
        {
            state = UserStates.正常;
        }
        return new UserService().ModifyUserState(userId, state);
    }
```

2. 会员状态管理表示层的设计与实现

1）基于管理端母版页 Admin.master 创建内容页 UserStateManage.aspx，并从工具箱拖入 GridView 控件至页面，在“GridView 任务”窗口中选择“编辑列”选项，对字段进行编辑。代码如下：

```
<asp:Button runat="server" ID="btnEnable" Text="启用用户" OnClick="btnEnable_Click" />
<asp:Button runat="server" ID="btnDisable" Text="禁止用户" OnClick="btnDisable_Click" />
<br /><br />
<asp:GridView ID="gvMain" runat="server" AutoGenerateColumns="False" CellPadding="0"
    CellSpacing="0" CssClass="data_table" DataKeyNames="Id" AllowPaging="True"
    OnPageIndexChanging="gv_PageIndexChanging" PageSize="5"
    OnRowCommand="gvMain_RowCommand">
    <Columns>
        <asp:TemplateField HeaderText="选择">
            <ItemTemplate><asp:CheckBox runat="server" ID="chkSelect" /></ItemTemplate>
            <ItemStyle HorizontalAlign="Center" />
            <HeaderTemplate>
                <input id="cbAll" type="checkbox" onclick="GetAllCheckBox(this)" />
            </HeaderTemplate>
        </asp:TemplateField>
        <asp:BoundField DataField="LoginId" HeaderText="用户名" SortExpression=
"LoginId" />
        <asp:BoundField DataField="LoginPwd" HeaderText="密码" SortExpression=
"LoginPwd" />
        <asp:BoundField DataField="Address" HeaderText="地址" SortExpression="Address" />
        <asp:BoundField DataField="Phone" HeaderText="电话" SortExpression="Phone" />
        <asp:BoundField DataField="Name" HeaderText="姓名" SortExpression="Name" />
        <asp:BoundField DataField="Mail" HeaderText="电子邮件" SortExpression="Mail" />
        <asp:TemplateField HeaderText="角色">
            <ItemTemplate><%# Eval("UserRole.Name") %></ItemTemplate>
        </asp:TemplateField>
        <asp:TemplateField HeaderText="状态">
            <ItemTemplate>
                <asp:Label ID="lblId" runat="server" Visible="false" Text='<%# Eval
("Id") %>'>
```

```
                    </asp:Label>
                    <asp:LinkButton runat="server" ID="lbtnStatus" CommandArgument=
                                    '<%# Eval("Id") %>'  CommandName="UpdateState"
                                    Text='<%# Eval("UserState.Name") %>'>
                    </asp:LinkButton>
                </ItemTemplate>
            </asp:TemplateField>
        </Columns>
        <PagerStyle CssClass="pages" />
        <RowStyle BackColor="#DDF5D9" />
        <SelectedRowStyle BackColor="#CE5D5A" Font-Bold="True" ForeColor="White" />
        <AlternatingRowStyle BackColor="White" />
    </asp:GridView>
```

这里的用户角色、状态和全选列使用模板列实现。

2）编程实现会员状态管理相关功能

在会员状态管理页的后置代码文件 UserStateManage.aspx.cs 中编程实现会员状态管理的功能，代码如下。

```
using BookShop.Models;
using BookShop.BLL;
public partial class Admin_UserStateManage : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            this.gvMain.DataSource = new UserManager().GetUsers();
            this.gvMain.DataBind();
        }
    }

    protected void btnEnable_Click(object sender, EventArgs e)
    {
        this.ChangeState(UserStates.正常);
    }
    protected void btnDisable_Click(object sender, EventArgs e)
    {
        this.ChangeState(UserStates.无效);
    }
    /// <summary>
    /// GridView 中按钮"单击"事件
    /// </summary>
    protected void gvMain_RowCommand(object sender, GridViewCommandEventArgs e)
    {
        switch (e.CommandName)
        {
            case "UpdateState":
```

```
                    UserManager manager = new UserManager();
                    manager.ModifyUserState(Convert.ToInt32(e.CommandArgument));
                    this.gvMain.DataSource = manager.GetUsers();
                    this.gvMain.DataBind();
                    break;
                default:
                    break;
            }
        }

        /// <summary>
        /// 分页索引改变时事件
        /// </summary>
        protected void gv_PageIndexChanging(object sender, GridViewPageEventArgs e)
        {
            this.gvMain.PageIndex = e.NewPageIndex;
            this.gvMain.DataSource = new UserManager().GetUsers();
            gvMain.DataBind();
        }

        /// <summary>
        /// 更改状态
        /// </summary>
        /// <param name="State"></param>
        private void ChangeState(UserStates state)
        {
            UserManager manager = new UserManager();
            ArrayList al = new ArrayList();
            for (int i = 0; i < this.gvMain.Rows.Count; i++)
            {
                CheckBox cb = (gvMain.Rows[i].FindControl("chkSelect")) as CheckBox;
                int id = Convert.ToInt32((gvMain.Rows[i].FindControl("lblId") as Label).Text);
                if (cb.Checked == true)
                {
                    manager.ModifyUserState(id, state);
                }
            }
            this.gvMain.DataSource = manager.GetUsers();
            this.gvMain.DataBind();
        }
    }
```

任务 9-2-2　实现“新知书店”管理端用户信息的更新

【任务描述】

实现“新知书店”管理端用户信息管理页面，显示用户信息，效果如图 9-10 所示。

单击图 9-10 中某条记录的“选择”按钮，进入该记录的编辑页面，实现用户信息的编辑更新功能，效果如图 9-11 所示。



图 9-10 “新知书店”管理端用户信息显示效果



图 9-11 “新知书店”管理端用户信息编辑效果

【任务实施】

1. 用户信息更新数据访问层与业务逻辑层的实现

页面加载时，要获取所有“正常”用户信息，数据访问层和业务逻辑层要能获取所有“正常”用户信息，对某用户信息进行更新时，能实现根据用户 Id 对用户表 Users 中的相应用户记录进行修改。

1) 用户信息更新数据访问层的实现

在 BookShopDAL 项目的 UserService.cs 类中，添加获取“正常”用户信息的方法 GetNormalUsers()，并返回一个用户列表对象集合，代码与任务 9-2-1 中获取所有会员信息的方法 GetUsers()除了 SQL 语句不同外，其他代码一样，SQL 语句代码如下：

```
string sql = "SELECT * FROM users WHERE userstateid = " + Convert.ToByte(UserStates.正常);
```

添加一个根据当前用户 Id 获取用户信息的方法 GetUserById(int Id)，代码如下：

```
/// <summary>
/// 根据 id 查询单个用户
```

```
/// </summary>
/// <param name="id">用户 Id</param>
/// <returns>一个用户对象</returns>
public User GetUserById(int id)
{
    string sql = "SELECT * FROM Users WHERE Id = @Id";
    int userStateId;
    int userRoleId;
    using (SqlDataReader reader = SqlHelper.ExecuteReader(this.connection, CommandType.
Text, sql, new SqlParameter("@Id", id)))
    {
        if (reader.Read())
        {
            User user = new User();
            user.Id = (int)reader["Id"];
            user.LoginId = (string)reader["LoginId"];
            user.LoginPwd = (string)reader["LoginPwd"];
            user.Name = (string)reader["Name"];
            user.Address = (string)reader["Address"];
            user.Phone = (string)reader["Phone"];
            user.Mail = (string)reader["Mail"];
            userStateId = (int)reader["UserStateId"];  //FK
            userRoleId = (int)reader["UserRoleId"];    //FK
            reader.Close();
            user.UserState = new UserStateService().GetUserStateById(userStateId);
            user.UserRole = new UserRoleService().GetUserRoleById(userRoleId);
            return user;
        }
        else
        {
            return null;
        }
    }
}
```

添加一个根据用户对象修改用户信息的方法，代码如下：

```
/// <summary>
/// 修改用户信息
/// </summary>
/// <param name="user"></param>
public bool ModifyUser(User user)
{
    string sql = "UPDATE Users " + "SET " +
            "UserStateId = @UserStateId, " +                    //FK
            "UserRoleId = @UserRoleId, " +                      //FK
            "LoginId = @LoginId, " +
            "LoginPwd = @LoginPwd, " +
            "Name = @Name, " +
```

```
                "Address = @Address, " +
                "Phone = @Phone, " +
                "Mail = @Mail " + "WHERE Id = @Id";
        SqlParameter[] para = new SqlParameter[]
        {
            new SqlParameter("@Id", user.Id),
            new SqlParameter("@UserStateId", user.UserState.Id),        //FK
            new SqlParameter("@UserRoleId", user.UserRole.Id),          //FK
            new SqlParameter("@LoginId", user.LoginId),
            new SqlParameter("@LoginPwd", user.LoginPwd),
            new SqlParameter("@Name", user.Name),
            new SqlParameter("@Address", user.Address),
            new SqlParameter("@Phone", user.Phone),
            new SqlParameter("@Mail", user.Mail)
        };
        return SqlHelper.ExecuteNonQuery(this.connection, CommandType.Text, sql, para) > 0;
    }
```

2）会员状态管理业务逻辑层的实现

在 BookShopBLL 项目的类文件 UserManager.cs 中，添加业务逻辑处理代码，代码比较简单，在此省略。

2. 用户信息更新表示层的设计与实现

（1）基于管理端母版页 Admin.master 创建内容页 UserList.aspx，并从工具箱拖入 GridView 控件至页面，设置 GridView 控件的 ID 属性值为 gvMain，DataKeysNames 属性值为 Id，在“GridView 任务”窗口中选择“编辑列”选项，对字段进行编辑。设置完成后，代码如下：

```
<asp:GridView ID="gvMain" runat="server" AutoGenerateColumns="False"
    CellPadding="0" CssClass="data_table"  DataKeyNames="Id" AllowPaging="True"
    PageSize="3" OnRowDataBound="gvMain_RowDataBound"
    OnRowCommand="gvMain_RowCommand"
    OnPageIndexChanging="gv_PageIndexChanging">
    <Columns>
        <% -- 其他代码省略 -- %>
        <asp:CommandField HeaderText="操作" ShowSelectButton="True" />
        <% -- 其他代码省略 -- %>
    </Columns>
        <% -- 其他代码省略 -- %>
</asp:GridView>
```

在用户显示页后置代码文件 UserList.aspx.cs 中编写单击某一条记录的“选择”按钮事件方法代码如下：

```
/// <summary>
/// 单击某一条记录的"选择"按钮事件方法
```

```
/// </summary>
protected void gvMain_SelectedIndexChanging(object sender, GridViewSelectEventArgs e)
{
    int index = e.NewSelectedIndex;              //获取新选择的行
    string key = this.gvMain.DataKeys[index].Value.ToString();
    Response.Redirect("UserEdit.aspx?Id=" + key);
}
```

（2）基于管理端母版页 Admin.master 创建内容页 UserEdit.aspx，用于更新从页面 UserList.aspx 传递过来的 Id 所对应的用户信息，UserEdit.aspx 页面代码如下：

```
<table cellspacing="1" cellpadding="3"  class ="table_edit">
    <tr>
        <th>用户 Id</th>
        <td><asp:Label ID ="lblLoginId" runat ="server" ></asp:Label></td>
    </tr>
    <tr>
        <th>姓名</th>
        <td>
            <asp:TextBox ID="txtName" runat="server" Text='<%# Bind("Name") %>'>
            </asp:TextBox>
            <asp:RequiredFieldValidator ID="valrName" runat="server" ControlToValidate
="txtName"  ErrorMessage="请填写姓名"></asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <th>电话</th>
        <td>
            <asp:TextBox ID="txtPhone" runat="server" Text='<%# Bind("Phone") %>'>
            </asp:TextBox>
            <asp:RequiredFieldValidator ControlToValidate="txtPhone" ID="valrPhone"
                runat="server" ErrorMessage="请填写电话"></asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator ID="valePhone" runat="server" ControlToValidate=
                 "txtPhone"  ErrorMessage="电话号码输入不正确" ValidationExpression=
                 "(\(\d{3}\)|\d{3}-)?\d{11}"></asp:RegularExpressionValidator>
        </td>
    </tr>
    <tr>
        <th>地址</th>
        <td>
            <asp:TextBox ID="txtAddress" runat="server" Text='<%# Bind("Address") %>'>
            </asp:TextBox>
            <asp:RequiredFieldValidator ID="valrAddress" runat="server" ControlToValidate=
                "txtAddress"  ErrorMessage="请输入地址"></asp:RequiredFieldValidator>
        </td>
    </tr>
    <tr>
        <th> E-Mail</th>
        <td>
```

```
            <asp:TextBox ID = "txtMail" runat = "server" Text = '<% # Bind("Mail") %>'>
            </asp:TextBox>
            <asp:RequiredFieldValidator ID = "valrMail" runat = "server" ControlToValidate
= "txtMail"
                ErrorMessage = "请输入 Email 地址"></asp:RequiredFieldValidator>
            <asp:RegularExpressionValidator ID = "valeMail" runat = "server"
                            ControlToValidate = "txtMail" ErrorMessage = "Email 地址不正确"
                            ValidationExpression = "\w+([-+.']\w+)*@\w+([-.]\w+)
*\.\w+([-.]\w+)*">
            </asp:RegularExpressionValidator>
        </td>
    </tr>
    <tr>
        <td colspan = "2">  <asp:Button ID = "bntSave" runat = "server" Text = "保 存"
            OnClick = "bntSave_Click" /></td>
    </tr>
</table>
```

注意：这里绑定字段的方法为 Bind("字段名")，还能回忆起它与 Eval("字段名")的区别吗？此外，还对提交的内容进行非空及合法性验证。

在用户信息更新页后置代码文件 UserEdit.aspx.cs 中编程实现用户信息绑定及更新功能，代码如下：

```
using BookShop.Models;
using BookShop.BLL;

public partial class Admin_UserEdit : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)//页面加载事件方法代码
    {
        if (!IsPostBack)
        {
            User user = new User();
            UserManager manager = new UserManager();
            user = manager.GetUserById(Convert.ToInt32(Request.QueryString["id"]));
            lblLoginId.Text = user.LoginId;
            txtName.Text = user.Name;
            txtMail.Text = user.Mail;
            txtPhone.Text = user.Phone;
            txtAddress.Text = user.Address;
        }
    }

    protected void bntSave_Click(object sender, EventArgs e)//单击"保存"按钮事件方法代码
    {
        User user = new User();
        UserManager manager = new UserManager();
        user = manager.GetUserById(Convert.ToInt32(Request.QueryString["id"]));
```

```
            user.Name = txtName.Text;
            user.Mail = txtMail.Text;
            user.Phone = txtPhone.Text;
            user.Address = txtAddress.Text;
            manager.ModifyUser(user);
            Response.Redirect("~/admin/UserList.aspx");
        }
    }
```

任务 9-2-3　实现“新知书店”管理端用户信息的删除

【任务描述】

在任务 9-2-2 的基础上，实现“新知书店”管理端用户信息的单选删除功能，效果如图 9-12 所示。单击一条记录的“删除”按钮，给出提示，如果确认删除则将当前记录删除，否则不执行删除操作。



图 9-12　“新知书店”管理端用户信息删除效果

【任务实施】

1. 用户信息删除数据访问层与业务逻辑层的实现

实现单选删除用户，数据访问层和业务逻辑层需要根据选中的用户记录的 ID 值执行删除操作。

1）用户信息删除数据访问层的实现

在 BookShopDAL 项目的 UserService.cs 类中，添加根据用户 ID 执行删除操作的 DeleteUserById(int id)方法，代码如下：

```
/// <summary>
/// 根据 ID 删除用户
/// </summary>
/// <param name="id"></param>
public bool DeleteUserById(int id)
{
```

```
        string sql = @"DELETE OrderBook WHERE OrderID IN(SELECT Orders.Id FROM Orders
                       INNER  JOIN Users ON Orders.UserId = Users.Id WHERE UserId = @Id)
                       DELETE Orders where UserId = @Id
                       DELETE Users WHERE Id  = @Id";
        SqlParameter[] para  =  new SqlParameter[]
        {
            new SqlParameter("@Id", id)
        };
        return SqlHelper.ExecuteNonQuery(this.connection, CommandType.Text, sql, para) > 0;
    }
```

DeleteUserById(int id)方法的 SQL 语句较为复杂，作用是当删除一个用户信息时，该用户在“新知书店”数据库中其他数据表中的相应信息均应被删除，读者需仔细研读。

2）用户信息删除业务逻辑层的实现

在 BookShopBLL 项目的类文件 UserManager.cs 中，添加业务逻辑处理代码，比较简单，在此省略代码。

2. 用户信息删除表示层的设计与实现

在页面 UserList.aspx 的基础上，添加“删除”模板列，代码如下：

```
<asp:GridView ID="gvMain" runat="server" AutoGenerateColumns="False"
    CellPadding="0" CssClass="data_table"  DataKeyNames="Id" AllowPaging="True"
    PageSize="3" OnRowDataBound="gvMain_RowDataBound"
    OnRowCommand="gvMain_RowCommand"
    OnPageIndexChanging="gv_PageIndexChanging">
    <Columns>
        <% -- 其他代码省略 -- %>
        <asp:TemplateField ShowHeader="False">
                <ItemTemplate>
                    <asp:LinkButton ID="lbtnDelete" runat="server"
                        CausesValidation="False" CommandName="Del"
                        CommandArgument='<%# Eval("Id") %>' Text="删除">
                  </asp:LinkButton>
                </ItemTemplate>
        </asp:TemplateField>
        <% -- 其他代码省略 -- %>
    </Columns>
        <% -- 其他代码省略 -- %>
</asp:GridView>
```

在用户显示页后置代码文件 UserList.aspx.cs 中给“删除”按钮注册 JavaScript 脚本，并编写单击某一条记录对应的“删除”按钮事件方法，代码如下：

```
/// <summary>
/// 绑定时,为删除增加确认对话框
/// </summary>
protected void gvMain_RowDataBound(object sender, GridViewRowEventArgs e)
```

```
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        LinkButton lb = e.Row.FindControl("lbtnDelete") as LinkButton;
        lb.Attributes.Add("onclick", "return confirm('删除用户会将与此用户相关的订单一起
删除,确认删除吗?')");
    }
}

/// <summary>
/// 单击某一条记录的"删除"按钮事件方法
/// </summary>
protected void gvMain_RowCommand(object sender, GridViewCommandEventArgs e)
{
    if (e.CommandName == "Del")
    {
        int id = Convert.ToInt32(e.CommandArgument);
        new UserManager().DeleteUserById(id);
        this.gvMain.DataSource = new UserManager().GetNormalUsers();
        this.gvMain.DataBind();
    }
}
```

任务 9-2-4　实现“新知书店”管理端图书信息的删除

【任务描述】

在任务 8-2-1 图书列表页 BookList.aspx 的基础上,为“新知书店”管理端图书信息显示页面添加多选删除功能,效果如图 9-13 所示。



图 9-13　“新知书店”管理端图书信息的多选删除效果

选择要删除的图书记录并单击“删除”按钮,弹出“确认删除吗?”信息提示框,如果确认删除,则选中的图书记录全部删除,否则不执行删除操作。

【任务实施】

1. 图书信息多选删除数据访问层与业务逻辑层的实现

实现多选删除图书信息，数据访问层和业务逻辑层需要根据选中的图书记录 ID 值执行删除操作，再者，“新知书店”具有在线销售功能，若某一图书信息已经从图书表 Books 中被删除，在图书订单表 OrderBook 中却还有该图书的订购信息，这是严重的逻辑错误。为了避免出现这种错误，要建立 Books 表与 OrderBook 表的主外键约束关系，在删除 Books 表中某一图书信息之前，要先删除 OrderBook 中对应的图书信息。

1）图书信息多选删除数据访问层的实现

在 BookShopDAL 项目中添加类文件 OrderService. cs，编写根据图书 ID 值从 OrderBook 表中执行删除操作的 DeleteOrderBookById(int id)方法，代码如下：

```
using BookShop.Models;
using System.Configuration;
namespace BookShop.DAL
{
    public class OrderService
    {
        string connection = ConfigurationManager.ConnectionStrings["BookShop"].ConnectionString;
        /// <summary>
        ///根据图书 Id 值在 OrderBook 表中执行图书信息相关删除操作
        /// </summary>
        /// <param name="id">图书 Id</param>
        public bool DeleteOrderBookById(int id)
        {
            string sql = "DELETE OrderBook WHERE BookID = @BookID";
            SqlParameter[] para = new SqlParameter[]
          {
             new SqlParameter("@BookID", id)
          };
            return SqlHelper.ExecuteNonQuery(this.connection, CommandType.Text, sql, para) > 0;
        }
    }
}
```

在 BookShopDAL 项目的 BookService. cs 类中，添加根据图书 ID 值从 Books 表执行删除图书信息的方法，代码如下：

```
/// <summary>
/// 根据图书 Id 值在 Books 表中执行删除图书信息
/// </summary>
/// <param name="id">图书 Id</param>
/// <returns></returns>
public bool DeleteBookById(int id)
{
    string sql = "DELETE Books WHERE Id = @Id";
```

```
        SqlParameter[ ] para = new SqlParameter[ ]
        {
          new SqlParameter("@Id", id)
        };
        return SqlHelper.ExecuteNonQuery(this.connection, CommandType.Text, sql, para) > 0;
    }
```

2）图书信息多选删除业务逻辑层的实现

在 BookShopBLL 项目的类文件 OrderManage.cs 和 BookManager.cs 中，添加业务逻辑处理代码，代码比较简单，此处省略。

2.图书信息多选删除表示层的设计与实现

（1）在页面 BookList.aspx 的基础上，添加“全选”模板列，代码如下：

```
<asp:GridView runat="server" ID="gvBooks" AutoGenerateColumns="False" AllowSorting="True"
    CellPadding="0" CssClass="data_table" OnRowDataBound="gvBooks_RowDataBound"
    OnPageIndexChanging="gvBooks_PageIndexChanging" AllowPaging="True"
    DataKeyNames="id"  OnSorting="gvBooks_Sorting" Width="98%">
    <Columns>
        <%--其他代码省略--%>
        <asp:TemplateField HeaderText="全选">
            <ItemTemplate>
                <asp:CheckBox runat="server" ID="chbSelect" />
            </ItemTemplate>
            <HeaderTemplate>
                <input type="checkbox" onclick="GetAllCheckBox(this)" />全选
            </HeaderTemplate>
            <ControlStyle Width="50px" />
        </asp:TemplateField>
    </Columns>
        <%--其他代码省略--%>
</asp:GridView>
```

（2）在页面 BookList.aspx 中添加“删除”按钮，代码如下：

```
<asp:Button ID="btnDel" runat="server" Text="删除" OnClick="btnDel_Click" />
```

（3）在页面 BookList.aspx 中添加实现“全选”功能的客户端 JS，代码如下：

```
<script language="javascript">
    function GetAllCheckBox(CheckAll) {
        var items = document.getElementsByTagName("input");
        for (i = 0; i < items.length; i++) {
            if (items[i].type == "checkbox") {
                items[i].checked = CheckAll.checked;
            }
        }
```

```
    }
</script>
```

(4) 在后置代码文件 UserList.aspx.cs 中给“删除”按钮注册 JavaScript 脚本，并编写单击“删除”按钮事件方法，代码如下：

```
protected void Page_Load(object sender, EventArgs e)            //页面加载事件方法
{
    if (!IsPostBack)
    {
        this.gvBooks.Attributes.Add("SortExpression", "Title");//给 GridView 添加排序字段
        this.gvBooks.Attributes.Add("SortDirection", "ASC");   //给 GridView 添加排序方式
        BooksDataBind();                                       //调用图书信息绑定方法
        //给"删除"按钮注册 JavaScript 脚本
        this.btnDel.Attributes.Add("onclick", "return confirm('确认删除吗?')");
    }
}

/// <summary>
/// 单击"删除"按钮事件方法
/// </summary>
protected void btnDel_Click(object sender, EventArgs e)
{
    foreach (GridViewRow gr in this.gvBooks.Rows)              //循环每一行
    {
        //获取模板列中的 CheckBox 控件对象
        CheckBox chk = gr.FindControl("chbSelect") as CheckBox;
        if (chk.Checked)
        {
            //获取主键
            string key = this.gvBooks.DataKeys[gr.RowIndex].Value.ToString();
            BookManager manager = new BookManager();
            OrderManage order = new OrderManage();
            //执行删除操作
            order.DeleteOrderBookByBookId(Convert.ToInt16(key));
            manager.DeleteBookById(Convert.ToInt16(key));
            BooksDataBind();                                   //删除后重新绑定图书信息
        }
    }
}
```

9.3 项目实训

1. 实现“博客系统”评论发表

【需求说明】

- 注册会员和匿名用户都可以使用此功能。

- 在文章详细信息页中提供发表评论的功能,将评论显示到文章下面,如图 9-14 所示。
- 可以匿名评论,也可以登录评论。
- 发表评论后,该篇文章的评论数加 1。

图 9-14 “博客系统”评论发表页效果(article.aspx)

2. 实现“博客系统”评论显示

【需求说明】

- 注册会员和匿名用户都可以使用此功能。
- 评论按内容列表显示,包括评论内容、评论人、评论时间。

(提示:用 DataList 控件实现评论内容的绑定。)

9.4 单元小结

GridView 是 ASP.NET 中最常用的数据绑定控件之一,它以表格的方式实现数据的展示,集成了显示、选择、编辑、分页和排序等功能,在 ASP.NET 的数据绑定控件中非常具有代表性。本单元讲述了 GridView 控件基于单元格的数据获取及更新,并阐述了 GridView 常用事件的处理和数据删除,其他数据绑定控件与 GridView 控件的使用大同小异,掌握了 GridView 控件的相关操作之后,使用其他数据绑定控件也就不存在障碍。本单元知识体系如图 9-15 所示。



图 9-15　数据绑定控件高级应用知识体系

9.5　单元练习题

一、选择题

1. 下列(　　)能够正确获取到 GridView 中的第三行第一列的值。

A. this. GridView1. Row[2]. Cell[0]. Text

B. this. GridView1. Row[3]. Cell[1]. Value

C. GridViewRow gvr=this. GridView1. Rows[3];
　String text=gvr. Cell[1]. Text;

D. GridViewRow gvr=this. GridView1. Rows[2];
　String text=gvr. Cell[0]. Text;

2. 以下关于 GridView 的模板列说法错误的是(　　)。

A. 使用模板列可以实现 HyperLinkField 的功能

B. 模板列在七个数据绑定列中是最灵活的

C. 在模板列中的 EditTemplate 中可以为输入控件添加验证

D. 模板列中可以添加 ASP.NET 控件，但不能添加 HTML

3. 以下(　　)事件在 GridView 完成每一行绑定数据时引发。

A. DataBound　　B. RowDataBound　　C. RowCreated　　D. DataBinding

4. 在 ASP.NET 中，要求 GridView 显示数据时，偶数行的文字为绿色，则下面代码中应该填写的是(　　)。

```
protected void gvPetShop ________(object sender, GridViewRowEventArgs e)
{
  if(e.Row.RowType == DataControlRowType.DataRow)
```

```
    {
      if(________ % 2!= 0)
      {
        e.Row.________.Add("style","color:green");
      }
    }
}
```

A. RowUpdating　　gvPetShop.currentIndex　　Styles

B. DataBound　　e.Row.RowIndex　　Attributes

C. RowDataBound　　gvPetShop.currentIndex　　Styles

D. RowDataBound　　e.Row.RowIndex　　Attributes

二、简答题

1. 简述实现 GridView 基于单元格更新的实现过程。

2. 为什么给 EditItemTemplate 中的 DropDownList 对象赋初值的时候不从 ItemTemplate 中的 Label 读取，而要在 EditItemTemplate 中放一个 HiddenFiled 对象？

单元10 指导学习："新知书店"购物功能的设计与实现

教学目标：

- 掌握数据绑定 Eval 和 Bind。
- 使用 GridView 展示多条数据。
- 掌握购物车的业务需求与增、删、改功能。

10.1 单元任务

任务 10-1-1 设计"新知书店"购物车商品实体类

【任务描述】

"新知书店"购物车商品实体需要保存图书的信息及该书的购买数量。

参考代码如下：

```
public class ShoppingItem
{
    private Book book;
    private int quantity;
    public Book Book              //采用图书类型作为 Book 属性的类型
    {
        get { return book; }
        set { book = value; }
    }
    public int Quantity           //图书购买数量
    {
        get { return quantity; }
        set { quantity = value; }
    }
    public ShoppingItem()         //构造方法
    {

    }
    public ShoppingItem(Book book, int quantity)
    {
        this.book = book;
```

```
            this.quantity = quantity;
        }
    }
```

任务 10-1-2　设计"新知书店"购物车类的业务逻辑

【任务描述】

所有的网上购物系统中，都实现了购物车功能，其过程与在商场将需要购买的物品装入购物车类似，如图 10-1 所示。

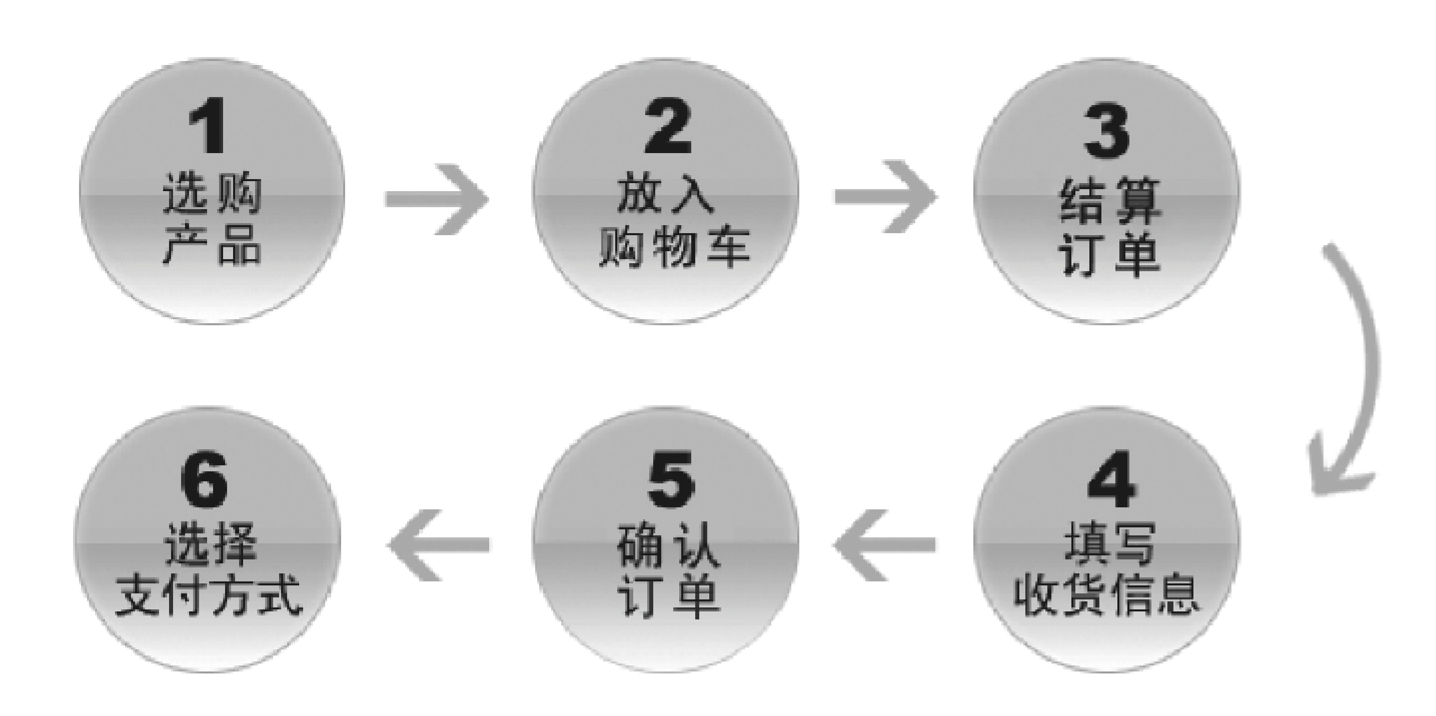


图 10-1　网上购物流程

从用户选取商品，到结算的过程都是在使用购物车进行数据的存储，那么如何实现购物车呢？

购物车一般具有以下功能：

- 把商品装入购物车。
- 显示购物车中的商品。
- 编辑购物车中的商品，主要是商品数量，并重新计算商品总价。
- 移除购物车中的商品。
- 清空购物车中的商品。

本书为突出购物车的关键业务逻辑，对购物流程中填写收货信息及支付等不予分析。

主要代码如下：

```
public class ShoppingManager
{
    private List<ShoppingItem> shoppingItems;
    private User user;
    public User User
    {
        get { return user; }
        set { user = value; }
    }

    /// <summary>
    /// 获得购物车书籍列表
```

```
/// </summary>
public List<ShoppingItem> ShoppingItems
{
    get
    {
        if (this.shoppingItems == null)
            this.shoppingItems = new List<ShoppingItem>();
        return this.shoppingItems;
    }
    set { this.shoppingItems = value; }
}

public ShoppingManager(object shoppingItems)
{
    this.ShoppingItems = shoppingItems as List<ShoppingItem>;
}

public ShoppingManager(object shoppingItems, object user)
{
    this.ShoppingItems = shoppingItems as List<ShoppingItem>;
    this.User = user as User;
}

/// <summary>
/// 添加书籍
/// </summary>
/// <param name="bookId"></param>
public void AddItem(int bookId)
{
    bool hadBuy = false;
    foreach (ShoppingItem item in this.ShoppingItems)
    {
        if (item.Book.Id == bookId)
        {
            hadBuy = true;
            item.Quantity += 1;
            break;
        }
    }
    if (!hadBuy)
    {
        Book book = new BookService().GetBookById(bookId);
        this.ShoppingItems.Add(new ShoppingItem(book, 1));
    }
}

/// <summary>
/// 删除书籍
/// </summary>
/// <param name="bookId"></param>
```

```
public void RemoveItem(int bookId)
{
    for (int i = 0; i < this.ShoppingItems.Count; i++)
    {
        if (this.ShoppingItems[i].Book.Id == bookId)
        {
            this.ShoppingItems.Remove(this.ShoppingItems[i]);
        }
    }
}

/// <summary>
/// 更新购买书籍数量
/// </summary>
/// <param name="bookId"></param>
/// <param name="quantity"></param>
public void UpdateQuantity(int bookId, int quantity)
{
    foreach (ShoppingItem item in this.ShoppingItems)
    {
        if (item.Book.Id == bookId)
        {
            item.Quantity = quantity;
            break;
        }
    }
}

/// <summary>
/// 由购物车生成订单
/// </summary>
public void MakeOrder()
{
    if (this.user != null && this.ShoppingItems.Count > 0)
        new OrderService().MakeOrder(this.ShoppingItems, this.user,true);
}

/// <summary>
/// 计算总价
/// </summary>
public decimal TotalPrice
{
    get
    {
        decimal totalPrice = 0;
        foreach (ShoppingItem item in this.ShoppingItems)
        {
            totalPrice += item.Quantity * item.Book.UnitPrice;
        }
        return totalPrice;
```

```
            }
        }
    }
```

这里要注意的是，因为最终的数据是通过订单模块存入数据库中的，所以只在业务逻辑层构建了 ShoppingManager 类，而在数据访问层则没有。

任务 10-1-3 实现"新知书店"购物车界面设计及显示

【任务描述】

实现"新知书店"购物车界面设计及绑定数据并显示的功能。购物车显示及编辑功能如图 10-2 所示（注意：在本书配套资源中，购物车页面 ShoppingCart. aspx 的正常显示需要从解决方案 BookShop 下的 BookDetail. aspx 页面中单击"购买"按钮进行链接；从页面 BookDetailsView. aspx 中单击"购买"按钮的方式没有实现，读者可以思考，并实现）。

部分重要代码参考如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    User user = Session["CurrentUser"] as User;
    if (user == null)
    {
        Page.RegisterClientScriptBlock("", "<script>alert('登录超时，请重新登录!');
document.location='Login.aspx';</script>");
        return;
    }

    if (!IsPostBack)                    //首次加载
    {
        if (Session["Cart"] != null)
        {
            ShoppingManager manager = new ShoppingManager(Session["Cart"], user);
            this.gvCart.DataSource = manager.ShoppingItems;
            this.gvCart.DataBind();
            this.ltrSalary.Text = string.Format("{0:F}", manager.TotalPrice);
        }
    }
}
```

运行购物车页面 ShoppingCart. aspx，效果如图 10-2 所示。

任务 10-1-4 实现"新知书店"购物车的增、删、改

【任务描述】

实现向"新知书店"购物车中添加书籍、修改购买数量、删除书籍及计算总价的功能。

难点提示：可以通过 GridView 实现向购物车添加记录，并显示；分别实现 GridView 中更新、删除和取消按钮的事件处理。



图 10-2 “新知书店”购物车效果

部分参考代码如下：

```
/// <summary>
/// 结算生成订单
/// </summary>
protected void btnCheckOut_Click(object sender, EventArgs e)
{
    ShoppingManager manager = new ShoppingManager(Session["Cart"], Session["
CurrentUser"]);
    if (manager.ShoppingItems.Count == 0)
    {
        Page.RegisterClientScriptBlock("", "<script>alert('您的购物车为空,请先将图书放
入购物车!');document.location='BookList.aspx';</script>");
        return;
    }
    if (manager.User == null)
    {
        Page.RegisterClientScriptBlock("", "<script>alert('登录超时,请重新登录!');
document.location='Login.aspx';</script>");
        return;
    }
    manager.MakeOrder();
    Session.Remove("Cart");
    Page.RegisterClientScriptBlock("", "<script>alert('结算成功,请等待审批订单!');
document.location='BookList.aspx';</script>");

}
```

```
/// <summary>
/// GridView 删除按钮处理事件
/// </summary>
protected void gvCart_RowDeleting(object sender, GridViewDeleteEventArgs e)
{
    ShoppingManager manager = new ShoppingManager(Session["Cart"]);
    Label lblBookId = this.gvCart.Rows[e.RowIndex].FindControl("lblBookId") as Label;
    int bookId = int.Parse(lblBookId.Text);
    manager.RemoveItem(bookId);
    Session["Cart"] = manager.ShoppingItems;
    this.gvCart.DataSource = manager.ShoppingItems;
    this.gvCart.DataBind();
    this.ltrSalary.Text = string.Format("{0:F}", manager.TotalPrice);
}

/// <summary>
/// GridView 取消按钮处理事件
/// </summary>
protected void gvCart_RowCancelingEdit(object sender, GridViewCancelEditEventArgs e)
{
    this.gvCart.EditIndex = -1;
    this.gvCart.DataSource = Session["Cart"] as List<ShoppingItem>;
    this.gvCart.DataBind();
}

/// <summary>
/// GridView 更新按钮处理事件
/// </summary>
protected void gvCart_RowUpdating(object sender, GridViewUpdateEventArgs e)
{
    ShoppingManager manager = new ShoppingManager(Session["Cart"]);
    foreach (GridViewRow dr in this.gvCart.Rows)
    {
        Label lblBookId = this.gvCart.Rows[e.RowIndex].FindControl("lblBookId") as Label;
        TextBox txtQuantity = this.gvCart.Rows[e.RowIndex].FindControl("txtQuantity") as
TextBox;
        int bookId = int.Parse(lblBookId.Text);
        int quantity = int.Parse(txtQuantity.Text);
        manager.UpdateQuantity(bookId, quantity);
    }

    Session["Cart"] = manager.ShoppingItems;
    this.gvCart.EditIndex = -1;
    this.gvCart.DataSource = manager.ShoppingItems;
    this.gvCart.DataBind();
    this.ltrSalary.Text = string.Format("{0:F}", manager.TotalPrice);
}

/// <summary>
/// GridView 数据绑定后激发的事件
```

```
/// </summary>
protected void gvCart_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
    {
        LinkButton lbtnDelete = e.Row.FindControl("lbtnDelete") as LinkButton;
        lbtnDelete.Attributes.Add("onclick", "return confirm('确定删除吗?')");
    }
}

/// <summary>
/// GridView 编辑按钮处理事件
/// </summary>
protected void gvCart_RowEditing(object sender, GridViewEditEventArgs e)
{
    this.gvCart.EditIndex = e.NewEditIndex;
    this.gvCart.DataSource = Session["Cart"] as List<ShoppingItem>;
    this.gvCart.DataBind();
}
```

10.2 单元小结

本单元以综合练习的方式要求分阶段完成“新知图书”前台购物车功能的设计与实现，依次从购物车的需求分析、设计与实现、购物车的界面设计与显示、购物车的编辑(增、删、改)给出了需求描述和操作提示。读者通过完成本单元的操作任务，能熟练掌握 Web 网站购物功能的实现方法和技术。

参 考 文 献

[1] 陈承欢. ASP.NET 网站开发实例教程. 北京：高等教育出版社，2011.
[2] 董义革，王萍，刘杨. ASP.NET 网站建设项目实战. 北京：北京邮电大学出版社，2013.
[3] 徐占鹏，苗彩霞. ASP.NET 程序设计. 北京：高等教育出版社，2013.
[4] 刘明彦，王超，肖宏启. ASP.NET 2.0 实用案例教程. 大连：大连理工大学出版社，2009.
[5] 代志勇，邵淑霞. ASP.NET 动态网站开发技术实践教程. 北京：中国铁道出版社，2011.
[6] 陈长喜，谢树龙. ASP.NET 程序设计基础教程. 第 2 版. 北京：清华大学出版社，2011.
[7] 宁云智，刘志成. ASP.NET 程序设计实例教程. 第 2 版. 北京：人民邮电出版社，2011.
[8] 杨玥，汤秋艳. Web 程序设计：ASP.NET(项目教学版). 北京：清华大学出版社，2012.